Real-Time Stereo Matching System for High Resolution Images on GPU

著者（英）	Chang Qiong
year	2019
その他のタイトル	GPUを用いた高解像度画像の実時間ステレオマッチングシステムの研究
学位授与大学	筑波大学 (University of Tsukuba)
学位授与年度	2018
報告番号	12102甲第9008号
URL	http://doi.org/10.15068/00156298

Real-Time Stereo Matching System for High Resolution Images on GPU

March 2019

Qiong Chang

Real-Time Stereo Matching System for High Resolution Images on GPU

Graduate School of Science and Engineering,
University of Tsukuba

March 2019

Qiong Chang

Abstract

Real-time stereo vision is attractive in many areas such as outdoor mapping and navigation. As a popular accelerator in the image processing field, GPU is widely used for the studies of the stereo vision algorithms. Recently, with the improvement of camera hardware, the resolution of the images is greatly improved. On the other hand, the research on acceleration methods for the stereo matching is being developed slowly. Many algorithms has been developed expecting the performance improvement of processors, but, its improvement speed cannot meet the that of increase of image size. This research aims to contribute in this regard by proposing a general stereo matching acceleration method, faster than current algorithms, that can work for high resolution system in real-time on GPUs.

Through the study of principle of stereo matching and a variety of algorithms, we propose an acceleration method that reduces the amount of computation of the system by reducing the original image before the processing and interpolating properly after the processing, thereby realizing the real-time processing. First, we verify the feasibility of our method for high-resolution images through a basic algorithm. Then we combine it with two other different algorithms. Each work uses a new idea from the image processing algorithm or acceleration method based on GPU, which helps to improve the processing speed over other systems.

We evaluate the performance of our three systems by using the Middlebury [16] and KITTI [27] benchmarks. Both of the benchmarks provide many types of datasets and corresponding ground truths. Images in these datasets have different sizes, different maximum disparities, different brightness, different textures, and different occlusion relationship. All of these can help us evaluate the performance of systems more comprehensively. Additionally, we run our systems on various of GPU platforms to ensure the versatility of our method.

According to the evaluation results, our second system achieves real-time processing for images with 2888 $\times$1920 pixels and a maximum disparity of 760. At the same time, we also improve the accuracy of the original Multi-Block Matching (MBM) algorithm that was used in this system. It mainly thanks to the combination of the image resizing and a secondary matching, which we propose in this paper. In our third system, we also run a more complex algorithm on an embedded GPU and apply it to the real world. By analyzing the relationship between GPU memory bandwidth and data accessing, we find an effective way to hide latency and achieve the real-time processing.

The main contribution of this work is to present a GPU method to accelerate the stereo matching for the high resolution images. This method can be applied to various GPUs and even other platforms to improve processing speed while

maintaining a certain accuracy. By using this method, our stereo matching system achieve the fastest speed in the world, which can save more time to do other works.

Acknowledgments

First of all, special thanks are due to my thesis supervisor Professor Maruyama, who supervised my work by providing me with valuable comments on major outcomes of the work. This thesis could not have been possible without him.

Second, I extend my gratitude to professor Onishi, who belongs to the National Institute of Advanced Industrial Science and Technology, Social Intelligence Research Team for financing my study. The conducive environment that the department created during my study also deserves lots of appriciation.

Thrid, I would especially like to thank my family. My parents and my wife have been extremely supportive of me throughout this entire process and has make countless sacrifices to help me get to this point.

Finnaly, I would like to thank all my friends in Tsukuba and around for the great moments we spent together.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The aim of stereo vision sytems is to reconstruct the 3-D geometry of a scene from images taken by two separate cameras. Because the principle of stereo matching is similar to the human eyes, as a key technology, it can be widely used in 3D-reconstruction [22] [23], robot vision [24], self-driving cars [25] and mechanical parts inspection. However, because of the high computational complexity of stereo matching, many applications [49] [50] based on it usually match for low resolution images to keep running smoothly. Since the lower resolution represents the larger actual distance between adjacent pixels, the accuracy of the systems which use the low resolution images is not high. Hence, other devices such as rados have been widely used to reconstruct 3-D reconstruction.

Recently, thanks to the development of the hardware, it makes it possible to use high resolution images for fast stereo matching. Many acceleration systems based on GPUs, FPGAs and delicated hardware have been developed [1] [2]. All of them succeeded in real-time processing of high resolution images, but their accuracy is not good enough because they simplified their algorithms to reduce the amount of calculation. Therefore, it is necessary to optimize high matching accuracy algorithms so that they can achieve high processing speed for high resolution images by fully utilizing target hardware architectures while maintaining their original matching accuracy.

Many algorithms on GPU have been developed. Because thousands of cores which run at faster than 1GHz are mounted on GPU chips, drastic performance gain can be expected. However, the processing speed of the stereo vision by GPUs is usually slower than FPGAs such as [3]. This is caused by the various limitations of GPU memory accessing. On the other hand, due to the higher freedom of software programming, more sophisticated algorithms can be implemented on GPUs than FPGAs, and lower error rates can be achieved. In recent years, the performance of GPUs is being improved continuously, and it becomes necessary to study how much speed up is possible by fully optimizing stereo matching algorithms on

GPU by using several techniques that can hide the memory access limitations.

1.1 Stereo Matching

1.1.1 Stereo Matching Theory

Stereo matching is a technique aimed to infer depth from two images taken by two separate cameras. As shown in Fig.1.1, when the two cameras are calibrated properly, epipolar restriction can be used for the pixels in the two images (left and right) to match with each other. The matching of pixels is searched to reconstruct the 3-D geometry of the scene. In Fig.1.1, P represents an object, P_L and P_R are the corresponding points where P is projected on the two images. $P_L^{'}$ is the projection of P_L on the right image. f is the focal length of both cameras, and B is the distance between the two cameras. According to triangulation, if we know the disparity between the two points $P_L^{'}$ and P_R, the distance Z from object P to the baseline B of the two cameras can be calculated using the following equation:

$$Z = f\frac{B}{d}. \tag{1.1}$$

Therefore, the main job in the stereo matching is to calculate the disparity of the pixels in the two images correctly.

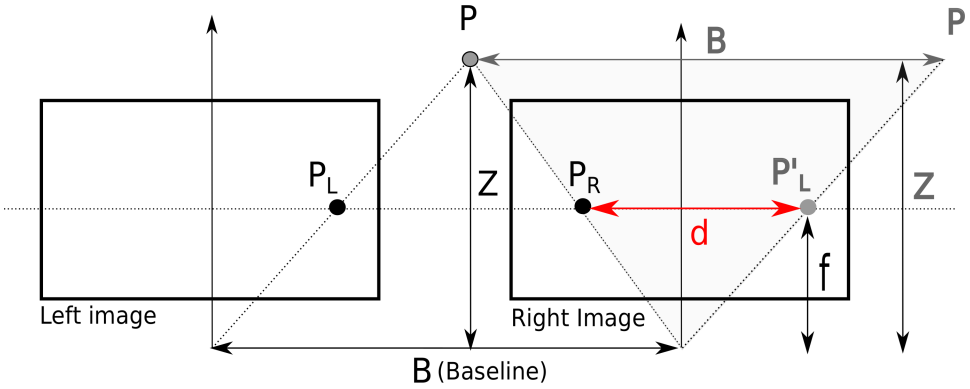

Figure 1.1: Stereo matching: Depth is calculated from the disparity of the corresponding pixels

To evaluate the matching result, a disparity map D_{map} is usually generated as shown in Fig.1.2. In this disparity map, the pixel values are represented by the disparity values of the corresponding pixels. The smaller the d, the darker the pixel, which means that the object is farther away from the cameras. $d = 0$ means that the object is at infinity. Fig.1.3 shows how to calculate a disparity map using

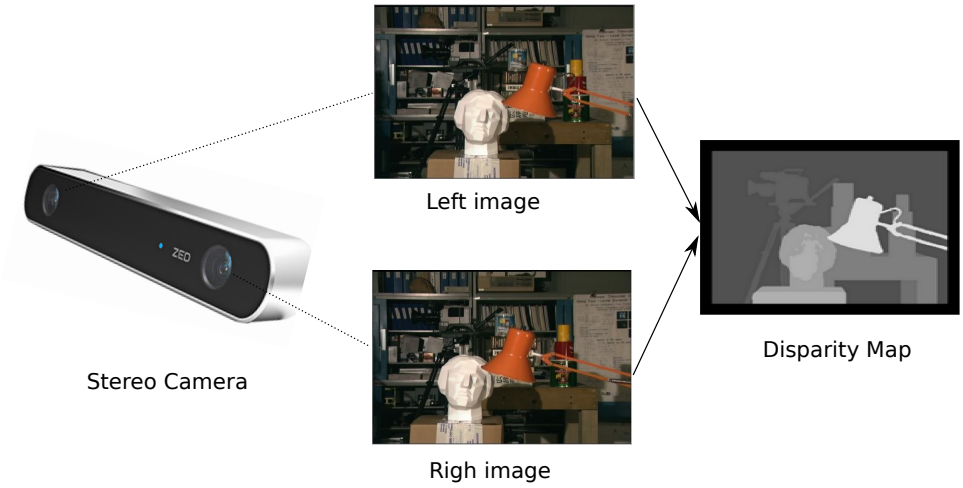

Figure 1.2: Generation of Disparity map: Disparty map is generated from two images taken by stereo camera

the left image as the base. A pixel in the left image $L(x, y)$ (or a window centered by $L(x, y)$) is compared with D pixels $R(x-d, y), d = [0, D-1]$ in the right image (or D windows centered by those pixels), and the most similar pixel to $L(x, y)$ is searched. D is the maximum disparity value. Suppose that $R(x-d, y)$ is the most similar to $L(x, y)$. Then, this means that $L(x, y)$ and $R(x-d, y)$ are the same point of an object.

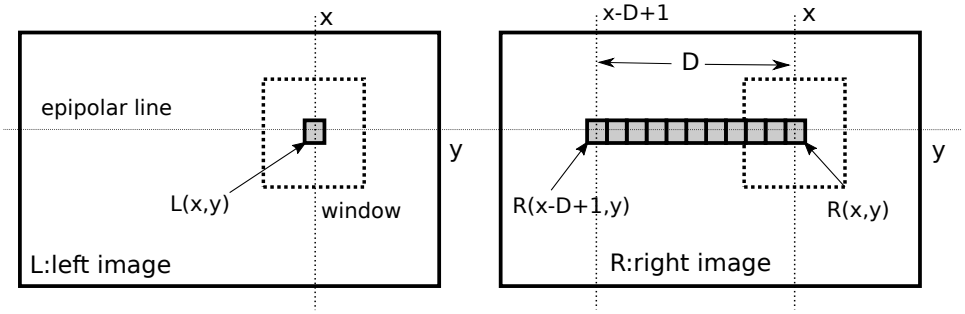

Figure 1.3: Local matching under the epipolar restriction

1.1.2 Stereo Matching Flow

Matching Cost Calculation

To detect whether two pixels $L(x, y)$ and $R(x-d, y)$ are the same object point is a very complicated process. The similarity between any two pixels is quantified

as a cost $C(x, y, d)$. As mentioned earlier, D costs are calculated for each pixel. These costs are calculated using the color difference, the absolute value of the difference, and/or the square differences and so on. Here, because the information obtained from a single pixel is limited, the matching costs obtained from only two pixels cannot reflect the true relationship between them. In general, the values of adjacent pixels are usually close, thus, it usually needs to combine the pixels in a window around them to determine if they are the same object point. This combination can be the sum of the color differences (SAD [15], SSD [56]), or a sequence of their size relationship (Census [57]), or cross-correlation between them (NCC [58]). The size of the window is usually small during this step.

Matching Cost Aggregation

Although the matching cost calculation relies on a small window to add information for each pixel, it is still a *point-to-point* matching, which cannot meet the high accuracy requirement. Therefore, higher accuracy matching unit based on *line* or *plane* is expected to improve the accuracy. Fortunately, for each pixel, the transformation can be easily performed by aggregating the costs $C(x, y, d)$ calculated earlier in a larger scope S. The new costs $C_S(x, y, d)$ are aggregated for each d to represents the similarity between the two regions, which are centered on the two matching pixels $L(x, y)$ and $R(x - d, y)$ respectively. Obviously, the aggregation greatly improves the accuracy of matching. Therefore, how to determine the scope S around each target pixel $L(x, y)$ and aggregate all the costs $C_S(x, y, d)$ are the most important issues, which are related to the overall amount of calculation and system precision.

For these issues, many algorithms have been proposed to date. According to the definition of the scope S, the algorithms can be categorized into two groups: local algorithms and global algorithms.

- Local Algorithm

  Only the local information around the target pixel $L(x, y)$ is used to decide the disparity $D_{map}(x, y)$. As shown in Fig.1.3, the scope S is defined as a small window (square window in general). The costs belong to this square window are aggregated based on different conditions, which are defined by different algorithms. Although the content and effect of these algorithms are different, the central idea of them is the same to try to find the pixels that belong to the same object. However, due to the limitations of local algorithm, it is difficult to handle repetitive patterns, perspective distortions and uniform regions.

- Global Algorithm

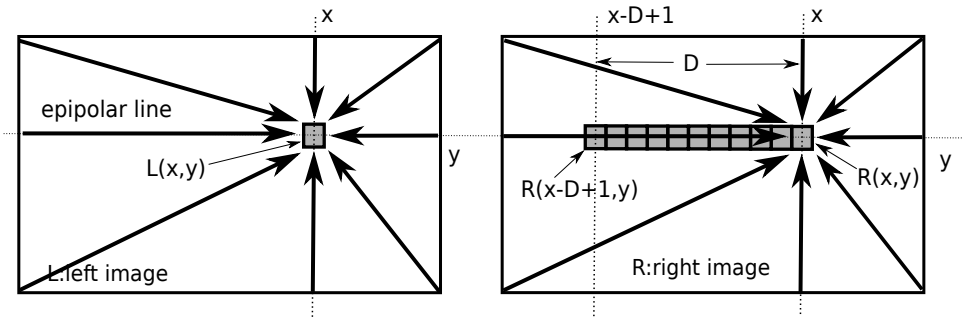

Figure 1.4: Global matching under the epipolar restriction

The disparities in D_{map} are decided considering the mutual effect of all pixels of L in the global algorithms as shown in Fig.1.4. Each pixel accepts the propagation of costs from surrounding pixels, and also spreads its own cost to the surrounding. For each target pixel $L(x, y)$, the degree of influence from other pixels is different, depending on the distance and color difference between them. Because the whole image is always handled for the matching of each pixel, it can usually achieve a good effect for the repetitive patterns and uniform regions. Accordingly, it needs more calculation and larger memory index range for each pixel.

Thus, it can be easily known that the system with global algorithms can achieve lower error rates than the system with local algorithms, but requires longer computation time. People can choose the appropriate algorithm according to their own needs.

After the cost aggregation, the disparity of each pixel is decided by Winner-Take-All (WTA). It means that for each pixel $L(x, y)$, the maximum(or minimum) cost $C(x, y, d)$ is chosen, and then the corresponding disparity d is set as the disparity of $L(x, y)$.

Disparity Refinement

Through the above steps, an initial disparity map D_{map} can be obtained. However, many outliers are contained for a variety of reasons. Among them, an important problem is caused by occlusion. As shown in Fig.1.5, when an object is taken by two cameras, some parts (the area in the red box) only appear in the left image but not in right side, depending on the positions and angles between the cameras and the object. The occlusion is the major source of outliers in the stereo vision systems. Some other outliers usually occur in the region of uniform or repetitive patterns, especially when the system is based on local algorithms. In such regions, the same cost are often obtained for different disparities, which leads to miss matching.

Figure 1.5: Occlusion areas cannot be matched correctly

The refinement is usually divided into two steps. One is the *GCP Detection* and the other one is *Non-GCP Filling*. The GCP is an abbreviation for *Global Control Points*, which refers to the reliable disparity, and those unreliable disparites, called outliers, are considered as Non-GCPs. The GCPs can be detected by checking whether the matching on both sides is consistent [59], or whether there is a contradiction between the matching results of any two pixels [60]. To fill the Non-GCPs, most of the methods use the surrounding disparities of GCPs. Some of them use the smallest disparity on the left and right sides [14], and some of them use the closest disparity [14]. Furthermore, some more complex algorithms correct the outliers by *Histogram Voting* [62] or calculating the *Least-squares Minimization* [63] of the surrounding disparities that belong to the same plane. In general, the more complex the algorithm, the higher the accuracy and the more time it takes. After that, in some cases, the disparity maps are simply refined by means of image filtering techniques like Median filtering, Bilateral filtering [64] without enforcing any constraint.

1.2 Background

1.2.1 GPUs

A graphics processing unit (GPU) is a specialized electronic circuit designed to rapidly manipulate and alter memory to accelerate the creation of images in a frame buffer intended for output to a display device. Modern GPUs are used in many areas like the personal computers, workstations, embedded systems and game consoles. Their highly parallel structure makes them more efficient than general-purpose CPUs for algorithms that process large blocks of data in parallel. Fig.1.6 shows the image of the architecture of Nvidia GPUs, which are widely used in the world. Although for different series of GPU, the number of streaming multi-processors (SMs) and the memory sizes are different, the hierarchy and the attribute of the memories have not any changed. Each SM has two types of memory: register memory and shared memory. The sizes of them are usually limited, but their access delay is very short. However, the data cached in the register memory are visible only to the thread that wrote them and last only for the lifetime of that thread, while the data cached in the shared memory are visible to all threads in the same SM and last for the duration of the kernel function which declared the threads. Each GPU also has the global memory, constant memory, and texture memory in the off-chip. The global memory is usually used to hold all data that are required for the processing, and the constant memory and texture memory are usually used to reduce the memory traffic to the global memory. By the reason of the long access delay to the off-chip memory, the most important technique to achieve high performance on GPU is how to cache a part of the data on the on-chip memory, and to hide the memory access delay to the global memory. The shared and the global memory have the restriction on the access to them. In the CUDA, which is an abstracted architecture of NVIDIA GPUs, 32 threads are managed as a set. When accessing the global memory, 32 words can be accessed in parallel if the 32 threads access to continuous 32 words on 32 word-boundary. Otherwise, the bank conflict happens, and several accesses to the global memory are issued. The shared memory consists of 32 banks, and the 32 words can be accessed in parallel if they are placed in different banks (the addresses of the 32 words do not need to be continuous).

1.2.2 Hardware Acceleration for Stereo Matching

Recently, many algorithms for stereo vision have been developed on different hardware. For global algorithms, [1] and [8] implemented the minimal spanning tree (MST) and dynamic programming (DP) on FPGA respectively. They achieved the real-time processing (30fps) for the high resolution images, but all of their dispar-

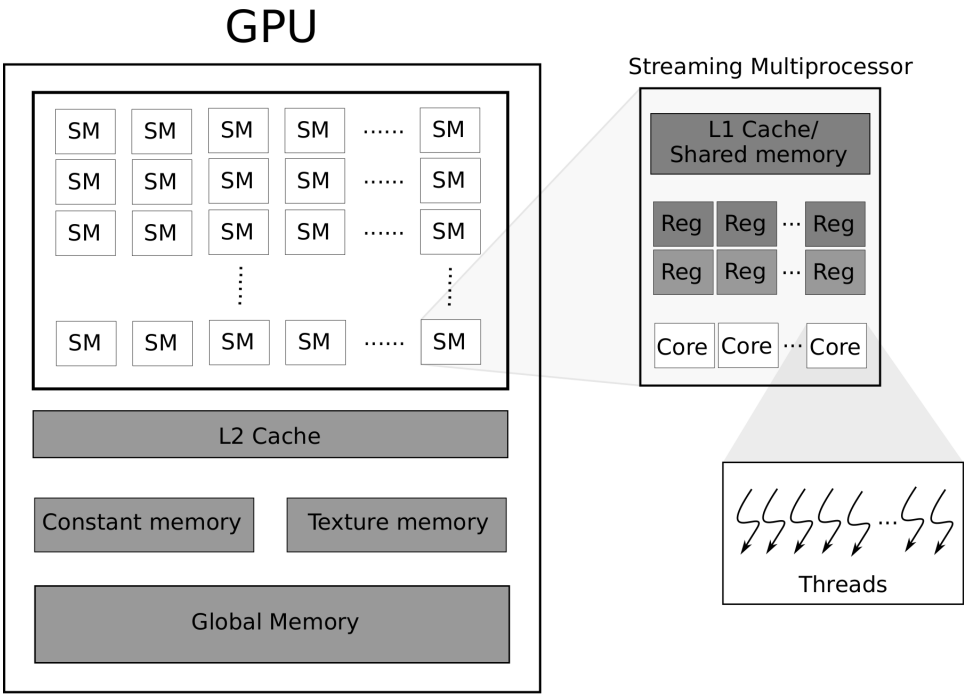

Figure 1.6: Nvidia GPU Architecture

ities are smaller than 64, which is not suitable to the modern requirements. [11] implemented a Semi-Global Matching (SGM) method on a low-end embedded GPU Tegra X1. According to the evaluation results in the KITTI Benchmark 2015, its error rate is 8.24% without the disparity refinement steps. However, it still has not achieved the real-time processing. [10] implemented a Recurrent Neural Network (RNN) aggregation method on a high-end GPU Geforce GTX Titan X in real-time. As the evaluation results of the KITTI Benchmark, its error rate is 6.34%, which is not very satisfied for a algorithm based on Neural Network. This is mainly caused by the simplification of the algorithms, in order to fit the hardware architecture.

For local algorithms, because only the local information around the target pixel is used to decide the disparity of the pixel, it is suitable for their acceleration on GPUs because the small and fast memory on GPU works efficiently owing to the high locality of the memory access. Cross-Based Aggregation (CROSS) is one of the most effective local algorithms. [13] and [12] implemented the CROSS on GPU and FPGA, and achieved real-time processing respectively. Although both of them achieved a low error rate, their input sizes are smaller than 640×480. [13] also runs their system for the high resolution images of Middlebury Benchmark. However, their speed is slower than 3fps for the large image set (2888×1920 pixels$\times$760 disparities). [4] implemented the Segmentation method on GPU for

high resolution images, which is used in the cost aggregation step. The same as [10] and [11], it also simplified the original algorithms in order to implement it on GPU, which resulted in decreased accuracy.

1.2.3 Problem Statement

As described previously, the existing stereo vision systems developed on hardware are facing limitations such as small image size, low processing speed or low accuracy. None of them can obtain a good balance. Some of these problems are due to the flaws of the algorithms themselves. Some are due to the fact that the algorithms cannot fit well to the hardware architecture. Additionally, some of them are caused by the low utilization of hardware. Because of all of these problems, the stereo matching technology cannot be used efficiently, and must be combined with other ancillary equipment. This not only increases the cost of the stereo vision system, but also limits the development of other related technologies.

1.3 Purpose of this Research

The main purpose of this research is to propose an method to accelerate the stereo matching for high resolution images by using GPU. This method is expected to solve the following problems: 1), It should be run in real-time. 2), The error rate should be at least maintained consistent with the original one running on CPU. 3), It must has high portability.

To achieve this purpose, first, we have to optimize the most computationally intensive part *Matching Cost Aggregation* by means of reducing the amount of computation or replacing it with other methods which can achieve the same effect. Generally, it is impossible to achieve real-time matching directly on the high resolution images, because the amount of image data is too large for the on-chip memory of the current GPU. We have to consider some new ideas to replace this original operation. Second, it is necessary to fully parallelize this part so that the architecture of GPU can be fully utilized. For local algorithms, they are easily processed by the GPU cores in parallel. However, we have to find the ways to keep the on-chip memory is not over-committed to ensure the high parallelism of threads. For the global algorithms, due to the high dependence between the pixels, we have to find the ways to increase the parallelism of the algorithms, while also minimizing the effect by the GPU limitation during the memory accessing.

Therefore, taking all the before mentioned requirements into consideration, we propose and implement three stereo vision systems with different algorithms. For these three algorithms, we accelerate them from different perspective, then evaluate them by using the popular benchmarks and the real world.

1.4 Thesis Outlines

This thesis is organized as follows. In chapter 2, we introduce the algorithms we researched for stereo matching. These algorithms are the base for all our GPU methods that are detailed starting from chapter 3 through the end of chapter 6. In chapter 3, we present our first acceleration method based on a local algorithm, which can reduce the amount of calculation significantly. In chapter 4, we present a new matching method based on the first one, which can maintain a high matching accuracy. In chapter 5, we present an acceleration method based on a global algorithm, and a method which can help us cover the latency of memory accessing effectively. Finally, in chapter 6 we summarize all the results achieved and discuss about them. Foll.owing this, in chapter 7, we mention the overall conclusions and talk about the future work.

Parts of this thesis have already been published. Most of the first method from chapter 2 has been published as a conference paper in [66]. Most of the second method from chapter 3 has been published as a journal paper in [65]. Finally, most of the third method from section 4.3 of chapter 4 has been published as a conference paper in [67].

Chapter 2

Algorithms for Stereo Matching

In Section 1.1.2, we have introduced the flow of stereo matching. There are many different algorithms, and they can be used in each step. Different combination of the algorithms leads to the disparity maps with different effects. In this section, we introduce the algorithms that were used in this thesis in detail.

2.1 Matching Cost Calculation

Here, we discuss three methods to calculate matching costs between two pixels.

2.1.1 Census Transform

In stereo matching, *Census transform* is widely used to determine the degree of similarity between two pixels. In the Census transform, the difference of brightness between the target pixel and its surrounding pixels are used. Equation 2.1 and 2.2 show the census equation and comparison process between the pixels.

$$S_L(x,y) = \underset{x',y' \in W}{\otimes} \xi(L(x,y), L(x',y')) \tag{2.1}$$

$$\xi(L(x,y), L(x',y')) = \begin{cases} 1, if \;\; L(x,y) < L(x',y'), \\ 0, if \;\; L(x,y) \geq L(x',y'), \end{cases} \tag{2.2}$$

For each target pixel $L(x,y)$, a square window W is defined around it. Then the relationships between the center pixel $L(x,y)$ and its surrounding pixels $L(x',y')$ is encoded in a bit string $S_L(x,y)$, which shows whether each pixel in the window is greater or less than the center pixel $L(x,y)$. By comparing the bit strings for the pixel $L(x,y)$ and $R(x-d,y)$ during the stereo matching, the matching cost

$C^L(x, y, d)$ which represents the degree of similarity between them can be given by Equation 2.3:

$$C^L(x, y, d) = H(S_L(x, y), S_R(x - d, y)). \tag{2.3}$$

Here, $S_L(x, y)$ and $S_R(x - d, y)$ are the census transforms (bit strings) of pixels $L(x, y)$ and $R(x - d, y)$. Function H represents the Hamming Distance between the two census transforms, and it is used to represent the matching cost of $L(x, y)$ in corresponding disparity d. The size of window W can be freely defined. In this thesis, we define it as 9×7.

One of the benefits of using the Census Transform is that the value of matching cost can be guaranteed as integer, which makes it possible to use faster logical operations instead of slower arithmetic operations.

2.1.2 Absolute Difference and Mini-Census Transform

Another simple cost calculation method is *Absolute Difference (AD)*. In the absolude difference, not like the census transform, the brightness of only the center pixels are used. This method is very simple, but it does not give good result because of the very limited information, and it is combined with other cost calculation method in general. In our approach, the absolute difference and mini-census are used for calculating the matching cost between two pixels. When the census transform is combined with other method, the size of the census window can be kept small (called *mini-census*). When the left image $L(x, y)$ is used as the base, the matching cost of the disparity $= d$ is given by

$$C^L(x, y, d) = C^L_{AD}(x, y, d) + C^L_{MC}(x, y, d). \tag{2.4}$$

$C^L_{AD}(x, y, d)$ is the cost by the absolute difference of the brightness of the two pixels, and given by

$$C^L_{AD}(x, y, d) = 1 - \exp(-\frac{|L(x, y) - R(x - d, y)|}{\lambda_{AD}}) \tag{2.5}$$

where λ_{AD} is a constant, In the same way, $C^L_{MC}(x, y, d)$, the cost by mini-census transform, is given by

$$C^L_{MC}(x, y, d) = 1 - \exp(-\frac{MC(S_L(x, y), S_R(x - d, y))}{\lambda_{MC}}) \tag{2.6}$$

where λ_{MC} is a constant, and Function MC represents the Hamming distance between the two mini-census transforms $S_L(x, y)$ and $S_R(x - d, y)$. Mini-census transform used in our approach is shown in Fig.2.1.

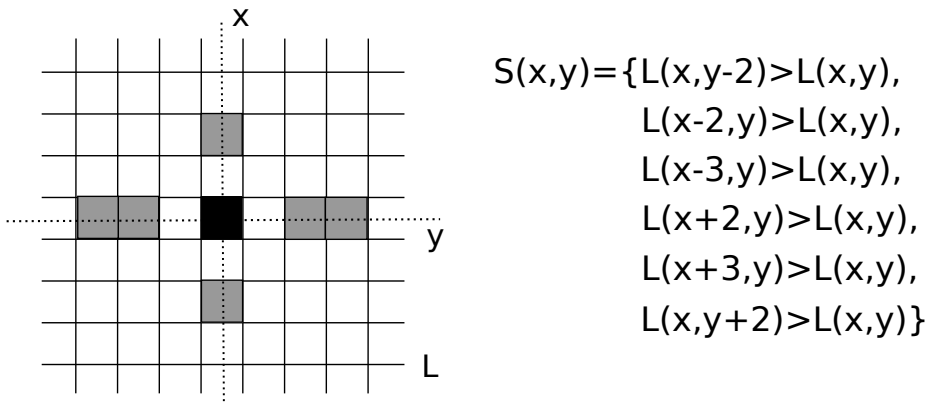

Figure 2.1: Mini-census Transform

The center pixel $L(x, y)$ is compared with its six neighbors, and a six bit string is generated as shown in Fig.2.1. This method is based on the hypothesis that the relative values of the brightness are kept in both images.

The advantage of this method is that the amount of calculation and data-width required are kept small, and this advantage is suitable for the GPUs with small on-chip memory.

2.1.3 Normalized Cross Correlation

The same as Census Transform, *Normalized Cross Correlation* (NCC) is also a method for matching two windows around the target pixels $L(x, y)$ and $R(x-d, y)$. The normalization within the window compensates differences in gain and bias. The cross correlation between $L(x, y)$ and $R(x - d, y)$ is given by:

$$C^L(x, y, d) = \frac{\sum_{x',y' \in W_p} (L(x', y') - \bar{L}(x, y))(R(x' - d, y') - \bar{R}(x - d, y))}{|W_p| \cdot \sigma_L(x, y) \cdot \sigma_R(x - d, y)} \tag{2.7}$$

where

$$\bar{L}(x, y) = \frac{1}{|W_p|} \sum_{x',y' \in W_p} L(x', y'), \tag{2.8}$$

and

$$\sigma_L(x, y) = \sqrt{\frac{1}{|W_p|} \sum_{x',y' \in W_p} (L(x', y') - \bar{L}(x, y))^2} \tag{2.9}$$

In this equation, W_p is the window used to calculate the NCC. $L(x', y')$ and $R(x' - d, y')$ are the pixels in each NCC window, while $\bar{L}(x, y)$ and $\bar{R}(x - d, y)$ are the averages of them. $\bar{R}(x-d, y)$ and $\sigma_R(x-d, y)$ are given in the same way as $\bar{L}(x, y)$ and $\sigma_L(x, y)$.

The main advantage of NCC is that it is less sensitive to linear changes of the intensity in the two matching images. Furthermore, The NCC cost is confined in range of $[-1, 1]$, which can ensure the cost value after the *Cost Aggregation* does not become too large.

2.2 Matching Cost Aggregation

As described in section 1.1.2, the *Matching Cost Aggregation* is the most important step during the stereo matching flow. Hence, we mainly focus on the acceleration methods for this step. In this thesis, we accelerated three algorithms on GPU, *Cross-Aggregation*, *Multi-Block Matching*, and *Domain Transformation*. The first two are local algorithms and the last one is a global algorithm. Each of them is widely used in many stereo vision systems.

2.2.1 Cross-Aggregation

In the Cross-Aggregation, the matching costs are aggregated as much as possible considering the similarity of the brightness of the pixels to compare the pixels as a block of the similar brightness. Fig.2.2 shows how the matching costs are aggregated. First, the matching costs $C^L(x, y, d)$ are aggregated along the x-axis.

$$CA_x^L(x, y, d) = \sum_{dx=-m}^{+n} C^L(x + dx, y, d) \tag{2.10}$$

Here, m and n are the number of the continuous pixels with the similar brightness to $L(x, y)$ ($|L(x, y) - L(x + dx, y)| < \delta$) on the left and right-side of $L(x, y)$. For example, in Fig.2.2, $m = 3$ and $n = 3$, because all pixels from $L(x - 3, y)$ to $L(x + 3, y)$ are similar to $L(x, y)$. Then, $CA_x^L(x, y, d)$ are aggregated along the y-axis as

$$CA^L(x, y, d) = \sum_{dy=-M}^{+N} CA_x^L(x, y + dy, d). \tag{2.11}$$

Here, M and N are the number of the continuous pixels with similar brightness to $L(x, y)$ on the upper and lower side of $L(x, y)$. In Fig.2.2, M is 4 and N is 4, because the pixels from $L(x, y - 4)$ to $L(x, y + 4)$ are similar to $L(x, y)$.

Then, d which minimizes or maximizes $CA^L(x, y, d)$ depending on the cost calculation method, is chosen as the disparity at $L(x, y)$, and disparity map $D_{map}^L(x, y)$

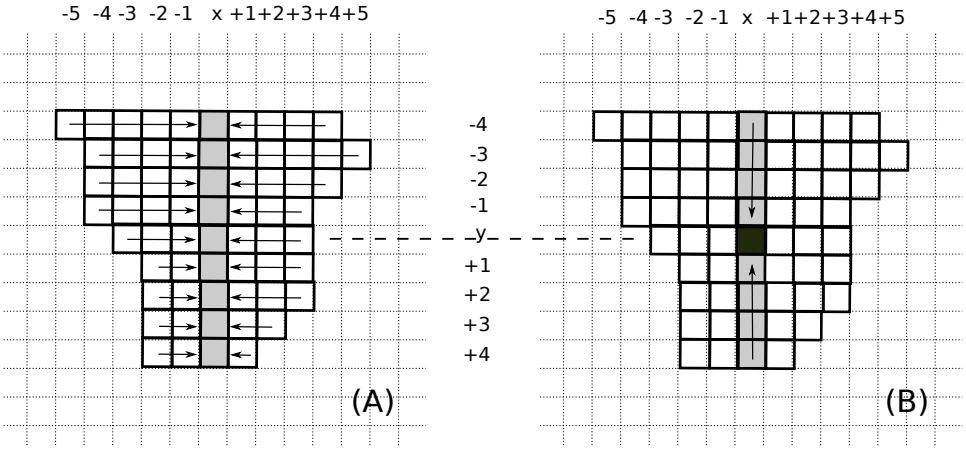

Figure 2.2: Cross-Aggregation

is obtained.

$$D^L_{map}(x,y) = \min \ or \max_{d \in D_{max}} CA^L(x,y,d). \tag{2.12}$$

By enlarging the range for summing up along the x- and y- axes, we can obtain more accurate disparities, though it requires more computation time.

2.2.2 Multi-Block Matching

The standard Block-Matching (BM) approach is widely used in the matching cost aggregation step. In this method, the matching costs of the pixels in a rectangular block centered at the target pixel are added, and the sum is used as the matching cost of the target pixel:

$$C^L_b(x,y,d) = \sum_{(x',y') \in b(x,y)} C^L(x',y',d). \tag{2.13}$$

where $b(x,y)$ is a rectangular block centered at (x,y).

However, the standard BM usually generates disparity maps with strong fattening. In order to obtain better disparity maps, [9] proposed the Multi-Block Matching (MBM) approach. It combines matching blocks of different shapes and sizes and has shown significant improvement compared to the standard BM. In the MBM, a multiplicative function of block set B is used to calculate the matching cost of each pixel:

$$C^L_{MBM}(x,y,d) = \prod_{b \in B} C^L_b(x,y,d). \tag{2.14}$$

where $C^L_b(x,y,d)$ is the matching cost calculated by standard BM for a block b. Fig.2.3(a) shows the cost aggregation in the standard BM. The matching costs of

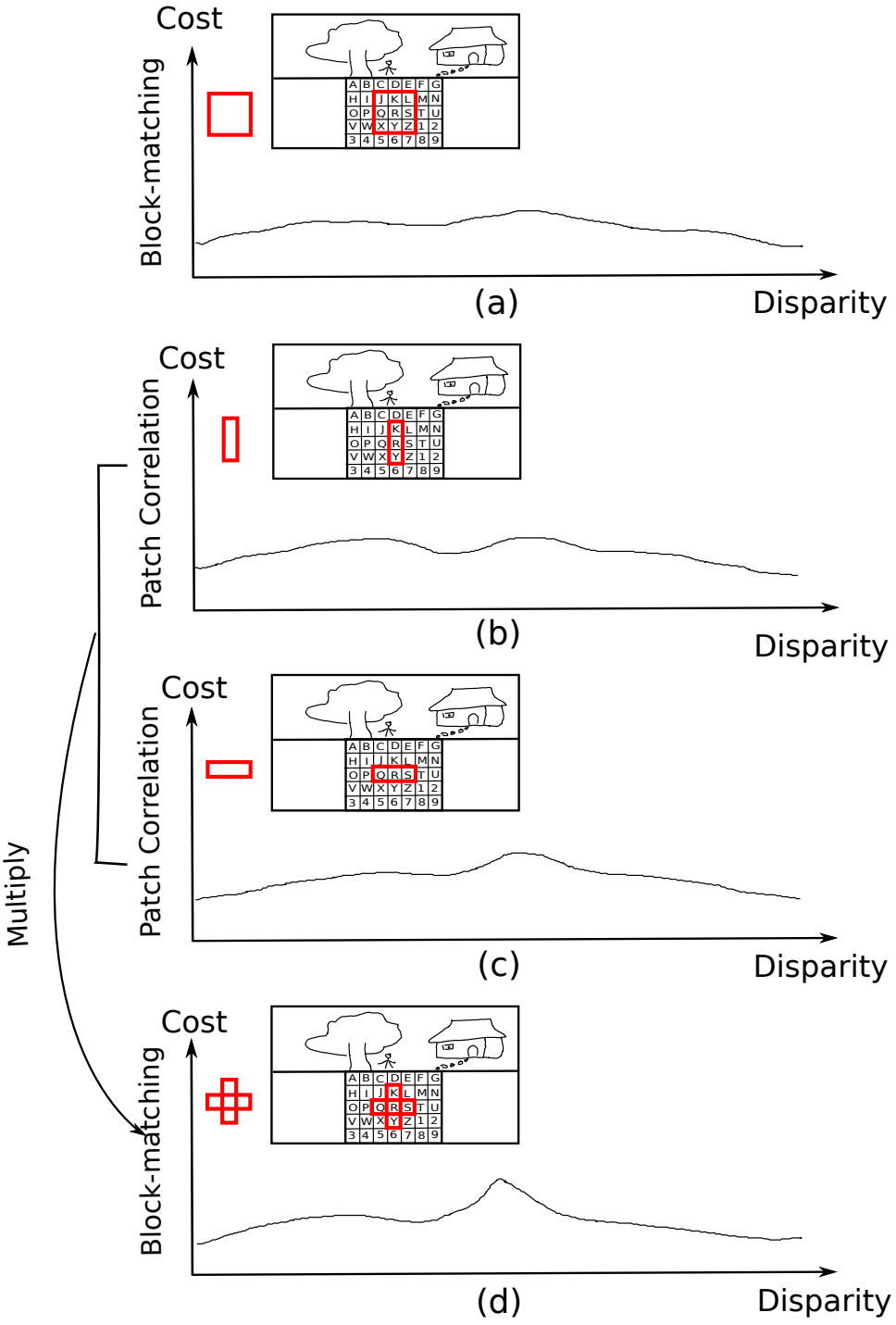

Figure 2.3: Multi-Block Matching

the pixels in the block are simply added. This approach often lacks the sensitivity because of the cost averaging in the large block. Fig.2.3(b) and (c) show the matching costs obtained when a vertical and horizontal block are used. In this example, as shown in Fig.2.3(c), a horizontal bock shows higher sensitivity because the texture changes sharply along the horizontal direction in the image. In this case, in the original BM, the sensitivity that could be obtained by the aggregation along only the x-axis is lost by the averaging of the pixels in the square block. However, it is preserved in the MBM in which the aggregated cost of each block is multiplied as the final matching cost (Fig.2.3(d)).

Finally, the disparity of each pixel $D^L_{map}(x, y)$ is decided by Winner-Take-All (WTA) as follows:

$$D^L_{map}(x, y) = \max \ or \min_{d \in D_{max}} C^L_{MBM}(x, y, d). \tag{2.15}$$

2.2.3 Domain Transformation

The main idea of Domain Transformation is roughly the same as Cross-Aggregation. Both of them are trying to aggregate the costs together in the same area (the area with similar brightness). The difference is that in Cross Aggregation, the maximum range of exploration around each center pixel is fixed, so it is called local algorithm. On the other hand, The Domain Transformation does not limit this range, but maximizes the cost aggregation from all four directions. Obviously, it is not cost-effective to aggregate all the costs belong to the same area from four directions for each pixel. [68] proposed a separate addition method which given by the following equation 2.16-2.19 to solve this problem.

$$C_L(x, y, d) = C(x, y, d) + C_L(x - 1, y, d) \cdot \exp\left(\frac{-|I(x, y) - I(x - 1, y)|}{\delta}\right) \tag{2.16}$$

$$C_R(x, y, d) = C_L(x, y, d) + C_R(x + 1, y, d) \cdot \exp\left(\frac{-|I(x, y) - I(x + 1, y)|}{\delta}\right) \tag{2.17}$$

$$C_D(x, y, d) = C_R(x, y, d) + C_D(x, y - 1, d) \cdot \exp\left(\frac{-|I(x, y) - I(x, y - 1)|}{\delta}\right) \tag{2.18}$$

$$C_U(x, y, d) = C_D(x, y, d) + C_U(x, y + 1, d) \cdot \exp\left(\frac{-|I(x, y) - I(x, y + 1)|}{\delta}\right) \tag{2.19}$$

According to these equations, the cost is continuously propagated along each direction, and the color difference between two adjacent pixels is used to weight the cost to distinguish different areas.

Although this method usually works well, the amount of calculation is still larger. To reduce the computational complexity, we reduce the aggregation along the y-axis to

$$C_U(x, y, d) = \sum_{\Delta y=-W}^{\Delta y=W} C_R(x, y + \Delta y, d). \tag{2.20}$$

In this way, the amount of calculation can be greatly reduced, and the problem caused by the excessive sensitivity of the image to the color can also be reduced.

2.3 Disparity Refinement

2.3.1 GCP Detection

Cross-Check

To pick out the outliers, *Cross-Check* has been widely used in many researches as an effective method. For performing the Cross-Check, two D_{map} are calculated [59]: once using the left image L as the base, and another using the right image R as the base. Then, the two disparity maps D_m^L and D_m^R can be generated and the reliable disparities (called the ground control points, or GCPs) can be judged as following:

$$D_{map}(x, y) = \begin{cases} D_{map}^L(x, y), & if \;\; D_{map}^L(x, y) = D_{map}^R(x - D_{map}^L(x, y), y) \\ 0, & if \;\; D_{map}^L(x, y) \neq D_{map}^R(x - D_{map}^L(x, y), y), \end{cases} \tag{2.21}$$

As shown in Fig.2.4, if any disparity $D_{map}^L(x, y)$ is equal to the corresponding disparity $D_{map}^R(x - D_{map}^L(x, y), y)$, it is assigned to the $D_{map}(x, y)$ as a GCP (the red matching), otherwise the $D_{map}(x, y)$ is set as 0 (the blue matching) and be filled according to the surrounding GCPs afterward.

Single Matching Phase

Single Matching Phase (SMP) [60] as a computationally efficient method, also works well. By checking whether two correspondences fall in the same point of the target image, the outliers can be detected. As shown in Fig.2.5, point C is matched to points A and B at the same time, which are on the same line. It means that at least one of them makes a mismatching. Hence, by comparing their matching costs, the correspondence with a smaller (or larger) cost is kept, then the

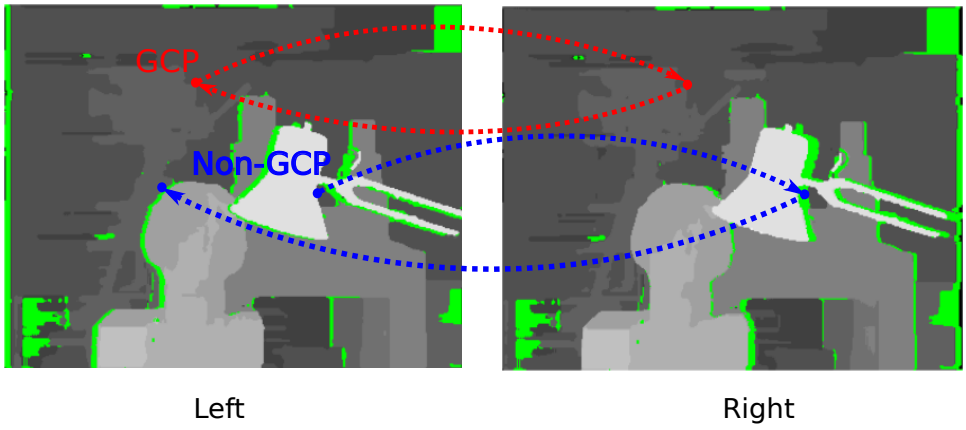

Figure 2.4: Cross-Check

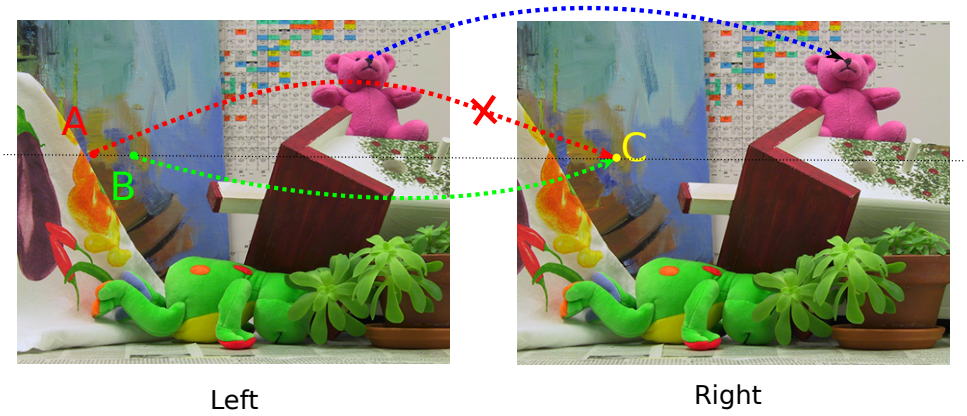

Figure 2.5: Single Matching Phase

other correspondence is discarded. Because this method only needs to calculate the disparity map of the target image, the calculation amount is only half of the *Cross-Check*. However, when the outliers are included on the same line, they may give the best matching score that leads to a mismatching.

2.3.2 Non-GCPs Filling

Sample Filling

To fill the disparity of non-GCPs, two simple methods are often used [14]. In both methods, for each non-GCP, the closest GCPs on the left and right hand-side along the x-axis are searched first. Then, in the first method, as shown in Fig.2.6(a), the closer GCP in the distance is chosen as the disparity of the non-GCP because the non-GCP and the closer GCP can be considered to belong to the same object with

a higher probability. In the second approach, as shown in Fig.2.6(b), the smaller disparity is chosen as the disparity of the non-GCP assuming that the non-GCP is caused by the occlusion. The disparity of the occluded region is smaller than that of the foreground object because the disparity of the closer object is larger, and the non-GCP should have a smaller disparity. Both of these two methods can be easily implemented on GPU because of their high parallelism.

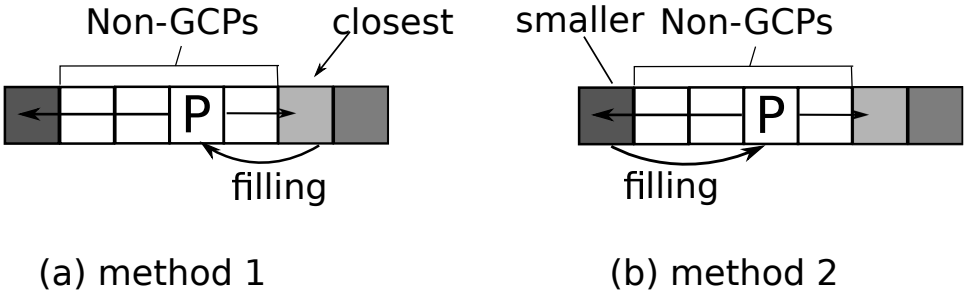

Figure 2.6: Non-GCPs Filling

Bilateral Filling

Here, we propose to use a bilateral estimation methods to fill the non-GCPs as following steps:

1. Suppose that the disparities of $L(x-i,y)$ and $L(x+j,y)$ are GCPs, and they are defined as $D_{map}(x-i,y)$ and $D_{map}(x+j,y)$.

2. If $|D_{map}(x-i,y) - D_{map}(x+j,y)| \leq T$, where T is the threshold for the difference of disparity, it can be considered that the disparity is changing continuously in this range, and $D_{map}(x,y)$ is filled as:

$$D_{map}(x-i,y) + i \cdot ((D_{map}(x-i,y) - D_{map}(x+j,y))/(i+j)). \quad (2.22)$$

3. If $|D_{map}(x-i,y) - D_{map}(x+j,y)| > T$, which means that the disparity changes rapidly in this range, it can be considered that an edge exists in this range. Thus $D_{map}(x,y)$ is chosen as the $D_{map}(x-i,y)$ if $L(x-i,y)$ is closer to $L(x,y)$ than $L(x+j,y)$ in color, and otherwise, $D_{map}(x,y)$ is chosen as $D_{map}(x+j,y)$.

With this approach, we can fill each non-GCP area properly, and an accurate disparity map can be expected.

Chapter 3

Approach for Reducing the Calculation

3.1 Algorithm Overview

We implemented a local search algorithm, which is an improved version of the algorithm on FPGA [12]. In this algorithm, AD (absolute difference) and mini-Census transform (Section 2.1.2) [69] are used to calculate the matching costs of the pixels in the two images, and they are aggregated along the x- and y-axes by using the *Cross-Aggregation* (Section 2.2.1) [70] [71]. In order to keep a high matching accuracy, *Cross-Check* is used to remove the effect by outliers.

In this method, we scale down the images to reduce the computational complexity. As shown in Fig.3.1, to reduce the computational complexity, we scale down the input images into a small size, and then scale up the disparity map to the original size. The images are scaled down to 1/4 by reducing the width and height by half, and the disparities are calculated on the scaled down images. The maximum disparity is also reduced to half, which means that the total computational complexity can be reduced to 1/8. This approach typically worsens the matching accuracy, but in our approach, an bilateral interpolation method is performed during the scaling up step, and a high matching accuracy can be maintained. This approach becomes possible because of the high quality of the high resolution images.

3.1.1 Processing Flow

Our algorithm consists of the following steps.

1. the two input images are gray-scaled
2. scaling down the two images

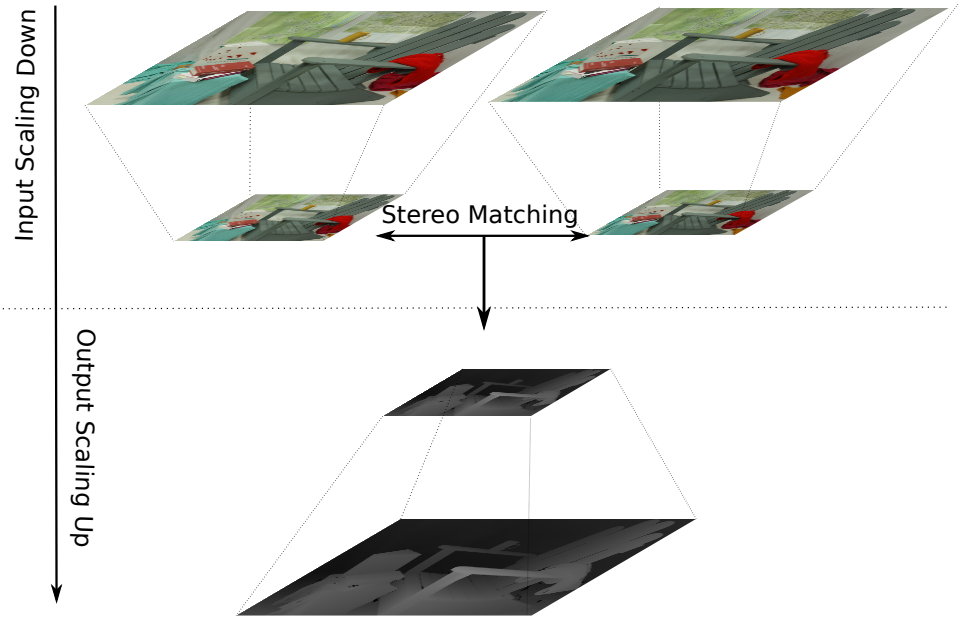

Figure 3.1: Image Scaling

3. calculating matching cost of each pixel using the AD-Census
4. cost aggregation along the x- and y-axes
5. generating two disparity maps
6. detecting GCPs (ground control pixels) by cross-checking the two disparity maps
7. refinement by a median filter and filling the non-GCPs by using a bilateral estimation method
8. scaling up the disparity map

3.1.2 Scaling Down

In order to reduce the computational complexity, the two images are scaled down linearly in both horizontal and vertical directions using the mean-pooling method. Here, take the left image as an example:

$$L(x,y) = \frac{1}{(2m+1)^2} \times \sum_{j=-m}^{m} \sum_{i=-m}^{m} L_{org}(K \cdot x + i, K \cdot y + j) \tag{3.1}$$

where $L_{org}(K \cdot x + i, K \cdot y + j)$ is the pixel in the original image, and K is the factor for the scaling down (in our implementation, $K = 2$). $L(x, y)$ is the pixel of the left scaled down image, and is smoothed by a mean-filter the size of which is $(2m + 1)^2$. By choosing the block size carefully, we can avoid the loss of the matching accuracy, and can improve the processing speed.

3.1.3 Calculation Reduction

As described in Section 2.1.2, when the right image $R(x, y)$ is used as the base, the matching cost is given as follows.

$$\begin{aligned} C^R(x, y, d) &= C^R_{AD}(x, y, d) + C^R_{MC}(x, y, d) \\ &= 1 - \exp(-\frac{|R(x, y) - L(x + d, y)|}{\lambda_{AD}}) \; + \\ &\quad 1 - \exp(-\frac{MC(S_R(x, y), S_L(x + d, y))}{\lambda_{MC}}) \\ &= C^L(x + d, y, d). \end{aligned} \tag{3.2}$$

This equation means that all $C^R(x, y, d)$ are already calculated when $C^L(x, y, d)$ are calculated, and $C^L(x, y, d)$ can be reused as $C^R(x - d, y, d)$.

3.2 Implementation on GPU

We have implemented the algorithm on Nvidia GTX780Ti. GTX780Ti has 15 streaming multi-processors (SMs). Each SM runs in parallel using 192 cores in it (2880 cores in total). For reducing the memory accesses to the global memory, the order of the calculation on GPU is different from the one described in the previous section.

- **Step1** transfer the input images onto the GPU and scale down them
- **Step2** compare the brightness of the pixels along the x-axis
- **Step3** calculate the matching costs and aggregate them along the x-axis
- **Step4** compare the brightness of the pixels along the y-axis
- **Step5** aggregate the cost along the y-axis, and generate two disparity maps
- **Step6** find GCPs by cross-checking
- **Step7** apply median filter to remove noises and estimate the disparity map using bilateral method

- **Step8** scale up the disparity map and transfer back to the CPU.

In the following discussion, $X_{org} \times Y_{org}$ is the image size (X_{org} is the width, and Y_{org} is the height), and $L_{org}[y][x]$ and $R_{org}[y][x]$ are the pixels in the left and right images. $L[y][x]$ and $R[y][x]$ are the pixels in the scaled-down images, and $X \times Y$ is the image size of them ($X = X_{org}/2$, $Y = Y_{org}/2$). Fig.3.2 shows the task assignment and input/output of each step. The details are discussed in the following subsections.

Step1

The inputs to this step are $L_{org}[y][x]$ and $R_{org}[y][x]$, and they are transferred onto the global memory of the GPU, and are processed in parallel using 15 SMs.

1. $Y_{org}/15$ lines of both images are assigned to each SM as shown in Fig.3.3.
2. X_{org} columns in the $Y_{org}/15$ lines are processed using X' ($X_{org} \leq X'$) threads in the SM (X' must be a multiple of 64 because of the reason described below). When $X' > X_{org}$, $X' - X_{org}$ threads work in the same way as the X_{org} threads, but generate no outputs.
3. When X_{org} is larger than the maximum number of the threads in one SM (1024), each thread processes more than one columns. In our implementation, because the resolution of the input images are greater than 1024, each thread processes 2 columns during the scaling-down step.

For each pixel $L_{org}(x_{even}, y_{even})$, both of the vertical and horizontal coordinates of which are even, and all of the surrounding pixels $L_{org}(x_{even} + dx, y_{even} + dy)$ ($dx \in [-1, 1]$, $dy \in [-1, 1]$) are added together. Then, as the pixel value of the scaled-image, the average of the summation is stored in the global memory.

Step2

For $Y/15$ pixels in one column (let the pixels be $L[y_b + k][x_b](k = 0, 14)$), each thread compares its pixel's value with its neighbors along the x-axis ($L[y_b+k][x_b+dx]$ ($dx = 1, W_x$) and $L[y_b + k][x_b - dx]$ ($dx = 1, W_x$)). The data type of $L[y][x]$ is $8b$ (unsigned char). Therefore, four continuous pixels are packed in one $32b$ word, and stored in the same memory bank of the shared memory. This means that these four continuous pixels can not be accessed in parallel owing to the memory access restriction of the shared memory. In order to avoid the bank conflict, $Thread_i$ processes column $(i/(X/4) + (i \times 4)\%X)$ as shown in Fig.3.3. In Fig.3.3, the first four pixels of each line ($L[y][0]$, $L[y][1]$, $L[y][2]$, $L[y][3]$) are stored in the first bank, and next four pixels ($L[y][4]$, $L[y][5]$, $L[y][6]$, $L[y][7]$) in the second bank. $Thread_0$

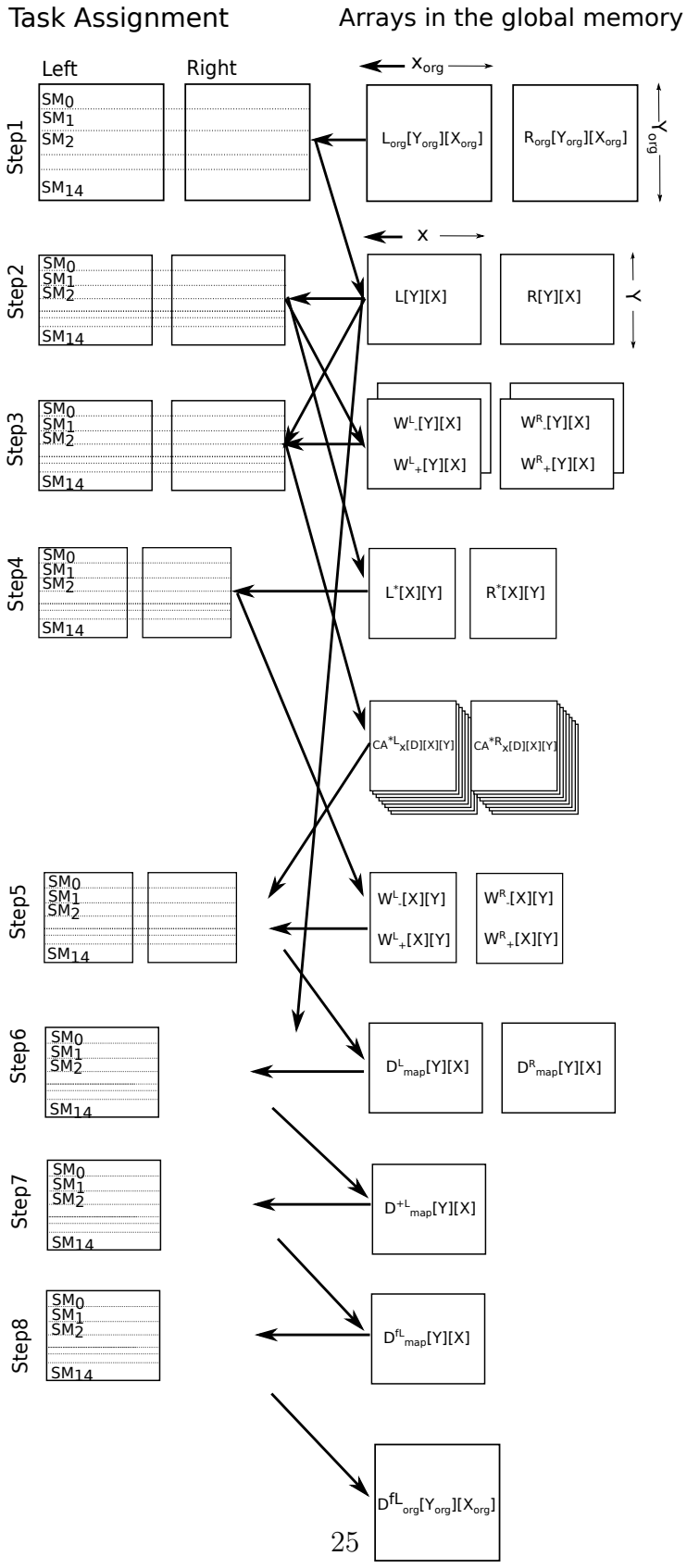

Figure 3.2: Task assignment of each step

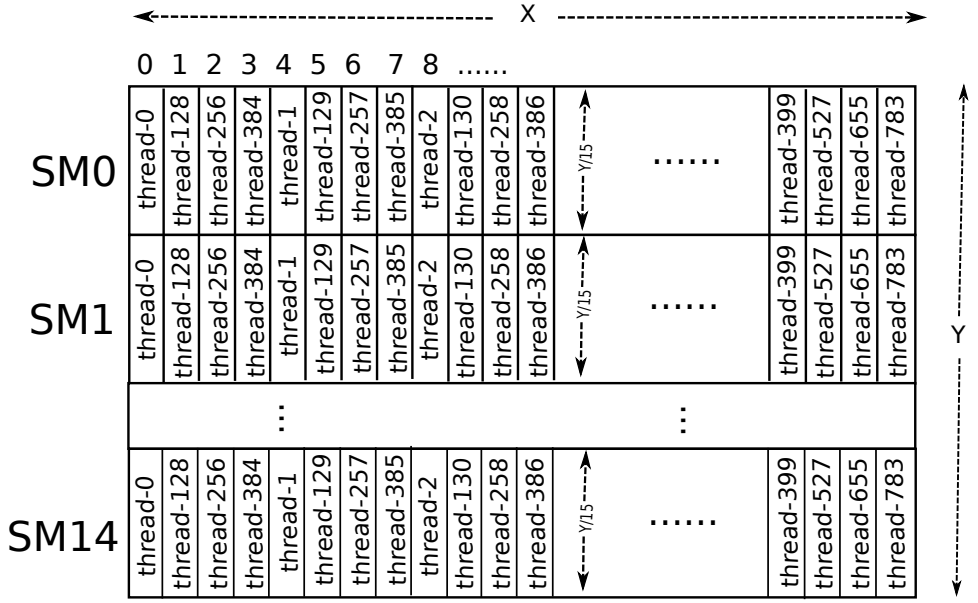

Figure 3.3: Mapping pixels to the threads

processes $Y/15$ pixels on $x = 0$ ($L[y_b + dy][0]$ ($dy = 0, 14$)) sequentially, and $Thread_1$ processes $Y/15$ pixels on $x = 4$ ($L[y_b + dy][4]$ ($dy = 0, 14$)) sequentially. By changing the order of the computation like this, bank conflict can be avoided. In our algorithm, all pixels can be processed independently, and the same results can be obtained regardless of the computation order.

In this method, four continuous pixels are stored in the same bank, and 16 threads are executed at the same time in CUDA. Therefore, X must be a multiple of 64 (4×16). When X is not the multiple of 64, larger X which is the multiple of 64 is chosen, and the computation results for the extended part are discarded.

The outputs of this step are two integer values for each pixel, which show how many pixels are similar to the center pixel to the plus/minus direction of the x-axis. These values for $L[y][x]$ are stored in $W^L_-[y][x]$ and $W^L_+[y][x]$, and those for $R[y][x]$ are stored in $W^R_-[y][x]$ and $W^R_+[y][x]$. The data width of these arrays is $32b$, and the direct access to these values causes no bank conflict. $L[y][x]$ and $R[y][x]$ are transposed here, and stored in $L^*[x][y]$ and $R^*[x][y]$ respectively.

Step3

In this step, first, two matching costs ($C^L(x, y, d)$ and $C^R(x, y, d)$) are calculated, and then, they are aggregated along the x-axis using the range information in $W^L_-[y][x]$, $W^L_+L[y][x]$, $W^R_-[y][x]$ and $W^R_+[y][x]$ to calculate $CA^L_x(x, y, d)$ and $CA^R_x(x, y, d)$. Here, actually, we do not need to calculate $C^R(x, y, d)$ as described in Section 3.1.3 because $C^L(x + d, y, d)$ can be used as $C^R(x, y, d)$. Therefore,

all SMs are used to calculate $C^L(x, y, d)$ as shown in Fig.3.2-*step*3, and each SM processes $Y/15$ lines as follows.

1. For each of the Y/15 lines, repeat the following steps.
2. Set $d = 0$.
3. Calculate $C^L(x, y, d)$ for all x in the current line. $C^L(x, y, d)$ is stored in $C[x]$ (an array in the shared memory). For this calculation, 3 lines of $L[y][x]$ and $R[y][x]$ are cached in the shared memory for calculating the mini-census transform, and they are gradually replaced by the next line as the calculation progresses.
4. Calculate $CA_x^L(x, y, d)$ as follows.
   (a) Set $CA_x[x] = C[x]$.
   (b) Add $C[x + dx]$ to $CA_x[x]$ starting from $dx = 1$ to the position given by $W_+^L[y][x]$.
   (c) Add $C[x - dx]$ to $CA_x[x]$ starting from $dx = 1$ to the position given by $W_-^L[y][x]$.
   (d) Store $CA_x[x]$ into $CA_x^{*L}[d][x][y]$ in the global memory (note that this array is transposed).
5. Calculate $CA_x^R(x, y, d)$ as follows.
   (a) Set $CA_x[x] = C[x + d]$.
   (b) Add $C[x + d + dx]$ to $CA_x[x]$ starting from $dx = 1$ to the position given by $W_+^R[y][x]$.
   (c) Add $C[x + d - dx]$ to $CA_x[x]$ starting from $dx = 1$ to the position given by $W_-^R[y][x]$.
   (d) Store $CA_x[x]$ into $CA_x^{*R}[d][x][y]$ in the global memory (note that this array is transposed).
6. Increment d if $d < D$, and go to step 3.

In this step, D arrays are stored in the global memory.

Step4

$L[y][x]$ and $R[y][x]$ have been transposed and stored as $L^*[x][y]$ and $R^*[x][y]$ in step2. By using these arrays, the brightness of the pixels are compared efficiently along the y-axis. In this case, the pixel data (for example $L^*[x][y]$) are compared

horizontally (parallel memory accesses are allowed only in this direction), and this means that $L^*[x][y]$ are compared with $L^*[x][y+dy]$ ($dy = -W_y, W_y$). The range of the similar pixels are stored in $W_-^{*L}[x][y]$ and $W_+^{*L}[x][y]$ for $L^*[x][y]$, and in $W_-^{*R}[x][y]$ and $W_+^{*R}[x][y]$ for $R^*[x][y]$. $L^*[x][y]$ and $R^*[x][y]$ are processed in parallel as shown in Fig.3.2-*step*4, and each SM processes $X/15$ columns. The Y lines in each column are assigned to Y threads in the same way shown in Fig.3.3 though x and y are transposed.

Step5

In this step (Fig.3.2-*step*5), $CA_x^{*L}[d][x][y]$ and $CA_x^{*R}[d][x][y]$ are aggregated along the y-axis in parallel using $W_-^{*L}[x][y]$, $W_+^{*L}[x][y]$, $W_-^{*R}[x][y]$ and $W_+^{*R}[x][y]$, and then $D_{map}^L[y][x]$ and $D_{map}^R[y][x]$ (disparity maps when $L[y][x]$ and $R[y][x]$ are used as the base) are also generated as follows.

1. Each SM processes $X/15$ columns.
2. Y threads in each SM processes Y pixels in each of the $X/15$ columns.
3. Each thread repeats the following steps (in the following, only the steps for the left image are shown).
   (a) Read $W_-^{*L}[x][y]$ and $W_+^{*L}[x][y]$ from the global memory.
   (b) Set $d = 0$.
   (c) $Min[y] = MAX\_VALUE$ and $D_{map}[y] = 0$.
   (d) Calculate $CA^L(x, y, d)$ as follow.
      i. Set $CA[y] = CA^{*L}[d][x][y]$.
      ii. Add $CA_x^{*L}[d][x][y+dy]$ to $CA[y]$ starting from $dy = 1$ to the position given by $W_+^{*L}[x][y]$.
      iii. Add $CA_x^{*L}[d][x][y-dy]$ to $CA[y]$ starting from $dy = 1$ to the position given by $W_-^{*L}[x][y]$.
   (e) If $CA[y] < Min[y]$ then $Min[y] = CA[y]$ and $D_{map}[y] = d$.
   (f) Increment d if $d < D$, and go to step 3(c).
   (g) Store $D_{map}[y]$ in $D_{map}^L[y][x]$ in the global memory (note that his array is re-transposed).

Step6

In this step (Fig.3.2-*step*6), $D_{map}^L[y][x]$ and $D_{map}^R[y][x]$ are read from the global memory line by line, and the condition for the GCP (described in Section 2.3.1)

is checked. In this step, each SM processes $Y/15$ lines. $Thread_x$ first accesses $D^L_{map}[y][x]$, and then $D^R_{map}[y][+k]$ if $D^L_{map}[y][x] = k$. k is different for each thread, and bank conflict happens in this step.

Step7

The median filter is applied using 15 SMs for the left image at first. Then, the bilateral estimation is used to fill the non-GCPs. If $D^L_{map}[y][x]$ is not a GCP, $thread_x$ scans $D^L_{map}[y][x]$ to the $+/-$ direction of the x-axis in order, and finds two GCPs (the GCPs closest on the left- and right-hand side). Then, the difference of the disparities of the two GCPs is calculated. If the difference is smaller than the threshold, the $D^L_{map}[y][x]$ is filled linearly according to the disparities of the two GCPs. If the different is larger than the threshold, the brightness of the two GCP are compared with the target pixel, and the disparity of the target pixel is replaced by the one which has similar brightness. Then, the improved disparity map $D^{+L}_{map}[y][x]$ is stored in the global memory.

Step8

Finally, the disparity map $D^{+L}_{map}[y][x]$ is scaled up by using the 15 SMs. In order to maintain a high accuracy, during the scaling-up along the x-axis, the bilateral estimation method described above is used again. On the other hand, the estimation along the y-axis is applied linearly. Then, the final disparity map $D^{fL_{org}}[y][x]$ is transferred back to the CPU.

sectionExperimental Results

The error rate and the processing speed are evaluated using Middlebury benchmark set [16].

In this evaluation, all parameters mentioned above affect the performance of the accuracy. According to our tuning results, we first set $\lambda_{AD} = 0.3$, $\lambda_{MC} = 2.3$ and $T = 3$ to ensure a good accuracy. In the cost aggregation step, lower error rates can be expected by adding more cost along the x- and y- axes, though it requires more computation time, and makes the system slower. The maximum range of the cost aggregation (W_x, W_y) can be changed when calculating D^L_{map} and D^R_{map}. By changing them, the different criteria are used for the left and right image, and the GCPs can be more reliable. Table 3.1 shows the error rate (%) when the cost aggregation range is changed. In Table 3.1, W_x are the maximum aggregation range along the x-axis for the left and right images, and W_y is the maximum aggregation range along the y-axis (W_y is common to the left and right images). As shown in Table 3.1, by enlarging W_y, the error rates can be improved when W_x is small. We have fixed $W_x = 21$ and $W_y = 31$. To our best knowledge, the accuracy of our system is higher than other real-time systems (like [6]) which are listed in Middlebury Benchmark [16]. Additionally, we also compared our error rates with

Table 3.1: Error rate when the cost aggregation range is changed (average error rate (%))

$W_L \backslash W_R$	$W_y = 9$	$W_y = 11$	$W_y = 15$	$W_y = 21$	$W_y = 27$	$W_y = 31$
$W_x = 5$	25.10	24.98	24.83	24.68	24.62	24.61
$W_x = 9$	24.67	24.55	24.4	24.3	24.26	24.26
$W_x = 21$	24.39	24.34	24.21	24.13	24.1	24.09
$W_x = 41$	24.53	24.38	24.29	24.21	24.19	24.17
$W_x = 61$	24.53	24.5	24.41	24.32	24.31	24.28
$W_x = 141$	24.6	24.55	24.48	24.42	24.38	24.28

Table 3.2: Execution Time For The Middlebury Benchmark Set (ms)

Image	**Size**	**Dmax**	**SD**	$W_{\pm}^{LR}$	$W_{\pm}^{*LR}$	**CA**	**Post**	**SU**	**Overall**
Adirondack(H)	1436×992	145	0.035	0.149	0.317	24.386	0.681	0.09	25.595
Pipes(H)	1482×994	128	0.038	0.135	0.289	31.24	0.744	0.09	32.536
Vintage(H)	1444×960	380	0.038	0.143	0.315	62.536	0.313	0.087	63.432

***Size**: $W_{\pm}^{LR} = 21$, $W_{\pm}^{*LR} = 31$, $TC = 13$ **Dmax**: Maximum Disparity SD: Scaling-Down. $W_{\pm}^{LR}$: Edge detection along the x-axis. $W_{\pm}^{*LR}$: Edge detection along the y-axis **CA**: Aggregation. **Post**: Cross_check, MedianFilter&Bilateral estimation. **SU**: Scaling-up. **Overall**: The overall time taken on GPU.*

that obtained using the original size image set, which are processed using larger window ranges $W_x = 41$ and $W_y = 61$. According to our evaluation, the error rate (Bad 2.0) of our system (24.09%) is higher than that by the original size images (32.82%). One of the reasons is that we didn't tuned the parameters λ_{AD} and λ_{MC} for the original image set. The other one is that in the scaling down images, some information that makes the matching difficult in the original size images, such as repetition of patterns and a serious of similar pixels, are discarded, and better matching becomes possible.

Fig.3.4 shows the results of our system for the two benchmark sets: Adirondack, Pipes and Vintage.

The processing speed of our system is almost proportional to the window size

Table 3.3: Comparison With High-Speed Stereo Vision Systems

System	**Size**	**Dmax**	**Hardware**	**Benchmark**	**FPS**	**MDE/s**
RT-FPGA [1]	1920×1680	60	Kintex 7	Middlebury v2	30	5806
FUZZY [2]	1280×1024	15	Cyclone II	Middlebury v2	76	1494
Low-Power [8]	1024×768	64	Virtex-7	Middlebury v2	30	1510
ETE [3]	1242×375	256	GTX TITAN X	KITTI 2015	29	3458
EmbeddedRT [11]	640×480	128	Tegra X1	KITTI 2012	81	3185
MassP [4]	1440×720	128	GPU	Middlebury v3	128	3981
Our system	1436×992	145	GTX 780 Ti	Middlebury v3	40	7849

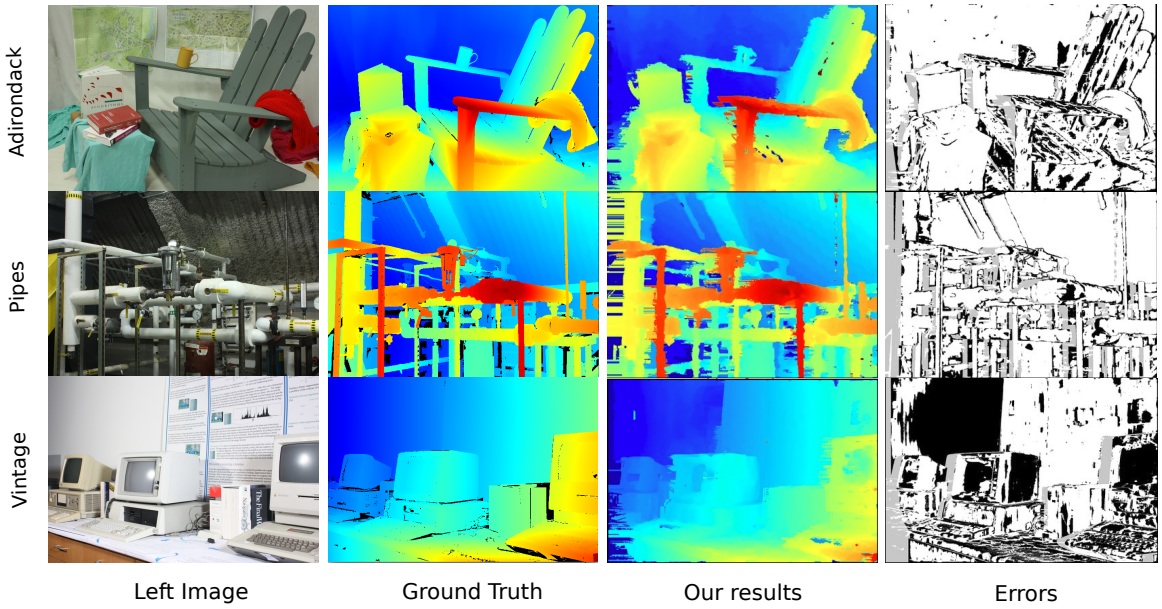

Figure 3.4: Processing results

and the maximum disparity. Therefore, the computation time for 'Vintage' becomes the slowest. Table 3.2 shows the processing speed of our system and its details. We ignore the time for CPU-GPU data transfers (less than 3% of the total elapsed time) since it can be overlapped with the computation. As shown in Table 3.2, most of the computation time is used for the cost calculation and its aggregation (CA). $W_{\pm}^{LR}$, $W_{\pm}^{*LR}$, $Post$, SD, SU show the computation time for finding the cost aggregation range along the x- and y- axes, cross checking, post-processing and the image scaling. Unfortunately, for the 'Vintage' set, its processing speed is 16fps due to the large disparity, and we cannot achieve the real-time processing.

Table 3.3 compares the processing speed of our system with other hardware systems. In Table 3.3, all of the systems achieved a real-time processing, but their target image size ($Size$) and disparity range ($Dmax$) are different. According to the mega disparity evaluation per second (MDE/S), it can be noted that our system is much faster than other systems.

Chapter 4

Approach for Accuracy Improvement

4.1 Algorithm Overview

In this chapter, we aim to construct a higher accuracy stereo system for the high resolution image set (2888×1920 pixels$\times 760$ disparities) in the Middlebury Benchmark. In order to achieve the balance of the processing speed and the accuracy, we continue the acceleration approach proposed in chapter 3. In addition, we added a secondary matching approach to improve the matching accuracy. The matching algorithm we used is the combination of NCC and MBM described in Section 2.1.3 and Section 2.2.2. The reason for choosing this combination is that it shows a good performance of processing speed and accuracy on CPU, and it can be easily combined with other methods to achieve a better performance, which makes our acceleration research more meaningful.

4.1.1 Processing Flow

In this system, the input images are processed as follows.

1. the two input images are gray-scaled,
2. the two images are scaled down,
3. the Normalized Cross-Correlation (NCC) is calculated as the matching cost of each pixel,
4. they are aggregated using three different shape and size blocks,
5. the Multi-Block Matching (MBM) is applied, and a disparity map is generated,

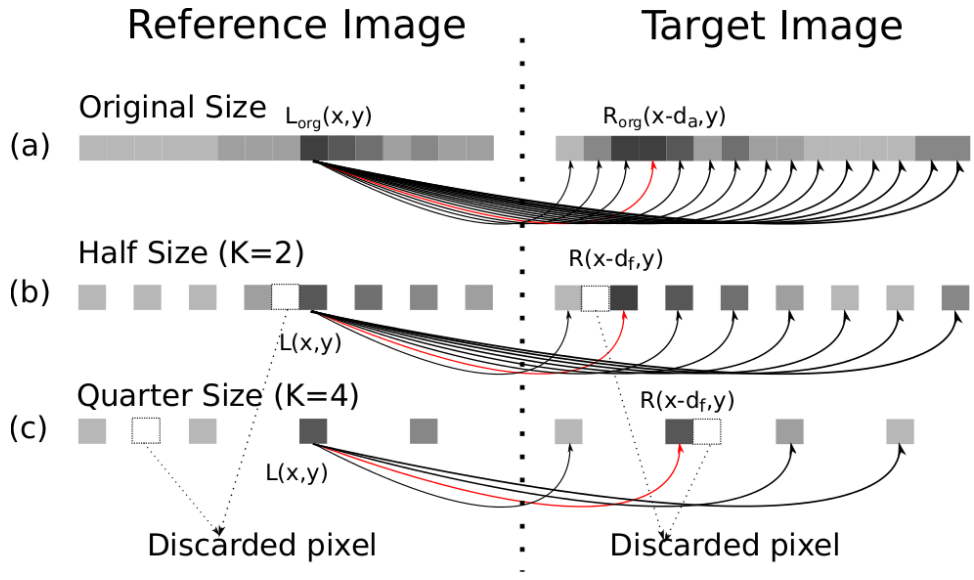

Figure 4.1: Stereo-Matching in Different Scale.

6. the secondary matching are performed, and then a disparity map is scaled up by the bilateral estimation,

7. step 4) to 6) are executed again to obtain two disparity maps using left and right image as the reference,

8. Ground control point (GCPs) are chosen using the two disparity maps, and

9. the disparity map is improved using the GCPs.

4.1.2 Secondary Matching

As described above, in order to reduce the computation complexity, the input images are scaled down to be processed. Fig.4.1 shows the stereo matching along one line on different scaling images. In Fig.4.1, all images on the left are the reference images, and those on the right are the target images. In Fig.4.1(a), the matching is performed on the original size images L_{org} and R_{org}, and an accurate disparity (d_a in this figure) can be found for each pixel $L_{org}(x, y)$. Fig.4.1(b) and (c) show the matching on the scaled down images. In these cases, only a fuzzy disparity (d_f) can be found due to the following factors:

1. During the scaling down step, part of the pixels in the target images R are discarded, illustrated by the blank space.

2. During the scaling up step, the disparity of each discarded pixel can only be estimated by using the disparities of the retained pixels, which may not be consistent with the truth value.

Accordingly, in order to improve the accuracy, we propose two methods to deal with these issues: a secondary matching method to find an accurate disparity for each pixel in the scaled down images, and a bilateral estimation method to fill in the disparities for the discarded pixels during the scaling up step.

As the analysis above, the accurate disparity d_a cannot be found by using the scaled down images. According to equation (3.1), K times the fuzzy disparity d_f is closest to the accurate disparity d_a and in most cases, d_a should be in $(K \cdot (d_f - 1), K \cdot (d_f + 1))$. Based on this assumption, in order to obtain a higher accuracy, after the MBM matching, a secondary stereo matching is performed.

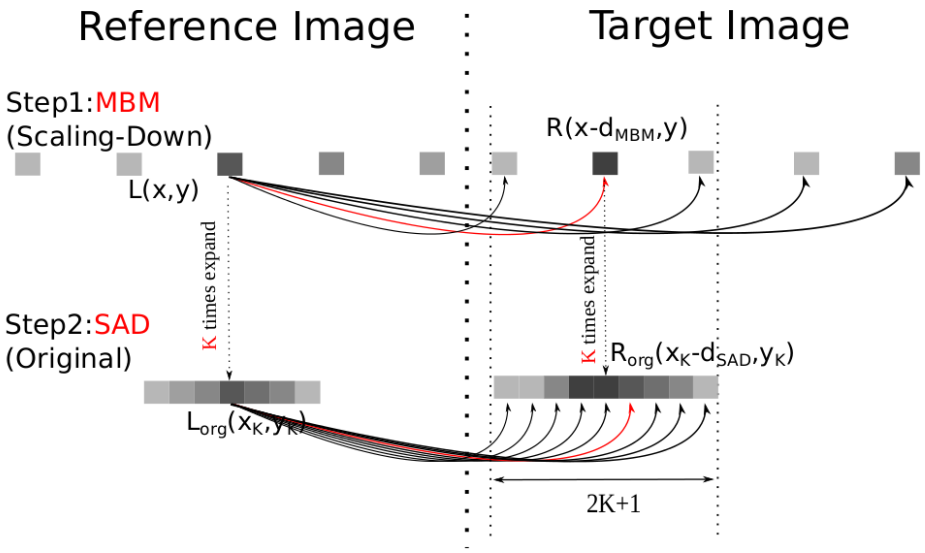

Figure 4.2: Secondary Matching.

As shown in Fig.4.2, for the pixel $L(x, y)$ in the scaled down image ($K = 4$), when the fuzzy disparity d_{MBM} is found by using the MBM, the corresponding pixel $L_{org}(x_k, y_k)$ in the original image ($x_k = K \cdot x$ and $y_k = K \cdot y$) is matched again with the original size target image R_{org}. Correspondingly, the discarded pixels in the range of $(K \cdot (d_{MBM} - 1), K \cdot (d_{MBM} + 1))$ are retrieved and used to find the accurate disparity. For this secondary matching, although many matching methods can be used such as MBM, in order to reduce the amount of computation, the simple matching method SAD [15] is chosen and the accurate disparities are

found as following:

$$C_{LSAD}(x_k, y_k, d) = \\ |S_q| \cdot I_{vol} - \sum_{x_k, y_k \in S_q} |L_{org}(x_k, y_k) - R_{org}(x_k - d, y_k)| \tag{4.1}$$

and

$$D_{LSAD}(x_k, y_k) = \max_d C_{LSAD}(x_k, y_k, d). \tag{4.2}$$

In (4.1), $C_{LSAD}(x_k, y_k, d)$ is the matching cost of pixel $L_{org}(x_k, y_k)$. S_q is the window used to calculate the cost of SAD, and I_{vol} is the maximum value in the gray-scale images (in our implementation, simply, 255 is used as I_{vol}). In (4.2), $D_{LSAD}(x_k, y_k)$ is the disparity of $L_{org}(x_k, y_k)$ that maximizes the value of $C_{LSAD}(x_k, y_k, d)$.

Fig.4.3 shows how the disparities are fine-tuned by using the secondary matching. First, Fig.4.3(a) shows a typical case of two matching costs for pixel $L(x, y)$; the matching cost by MBM and that by SAD. In this figure, d_{MBM} gives the best matching cost $C_{LMBM}(x, y, d_{MBM})$, and in the range of $d_{MBM} \pm 1$, the secondary matching costs $C_{LSAD}(x, y, d)$ by SAD are calculated ($K = 4$ in this example). The x-axis represents the disparities for the scaled down and original size image. The disparity $d_{\mathrm{MBM}} \pm 1$ in the scaled down image corresponds to $K \cdot (d_{\mathrm{MBM}} \pm 1)$ in the original size image. In order to ensure the robustness of this method, it is necessary to confirm that for each pixel, whether the result by SAD is consistent with that by MBM or not. If it is consistent, d_{SAD}, the disparity that maximizes $C_{LSAD}(x, y, d)$, should be between $(K \cdot (d_{MBM} - 1), K \cdot (d_{MBM} + 1))$, but should not be on $K \cdot (d_{MBM} - 1)$ or $K \cdot (d_{MBM} + 1)$, because $C_{MBM}(x, y, d_{MBM} \pm 1)$ is smaller than $C_{MBM}(x, y, d_{MBM})$. In Fig.4.3(a), this requirement is satisfied, but it is not in Fig.4.3(b). In the former case, the disparities are fine-tuned as described below, but in the later case, the matching cost by SAD is not used for fine-tuning, and d_{MBM} is used as the result.

In our approach, to fine-tune the disparities, a sub-pixel estimation method proposed by [14] is used. In [14], it is supposed that the curve of the matching costs is continuous on the disparities and each small fragment of the matching costs can be approximated by a quadratic function. In our system, using $C_{LMBM}(x, y, d_{MBM})$ and $C_{LMBM}(x, y, d_{LMBM} \pm 1)$, a quadratic function that goes through these three points is calculated, and the distance from d_{MBM} to the disparity that gives the peak to the quadratic function, Δd_{MBM}, is obtained. In the same way, a quadratic function that goes through $C_{LSAD}(x_k, y_k, d_{SAD})$ and $C_{LSAD}(x_k, y_k, d_{SAD} \pm 1)$ is calculated, and the distance from d_{SAD} to the disparity that gives the peak to the quadratic function, Δd_{SAD}, is obtained. Then, the disparity is fine-tuned as follows.

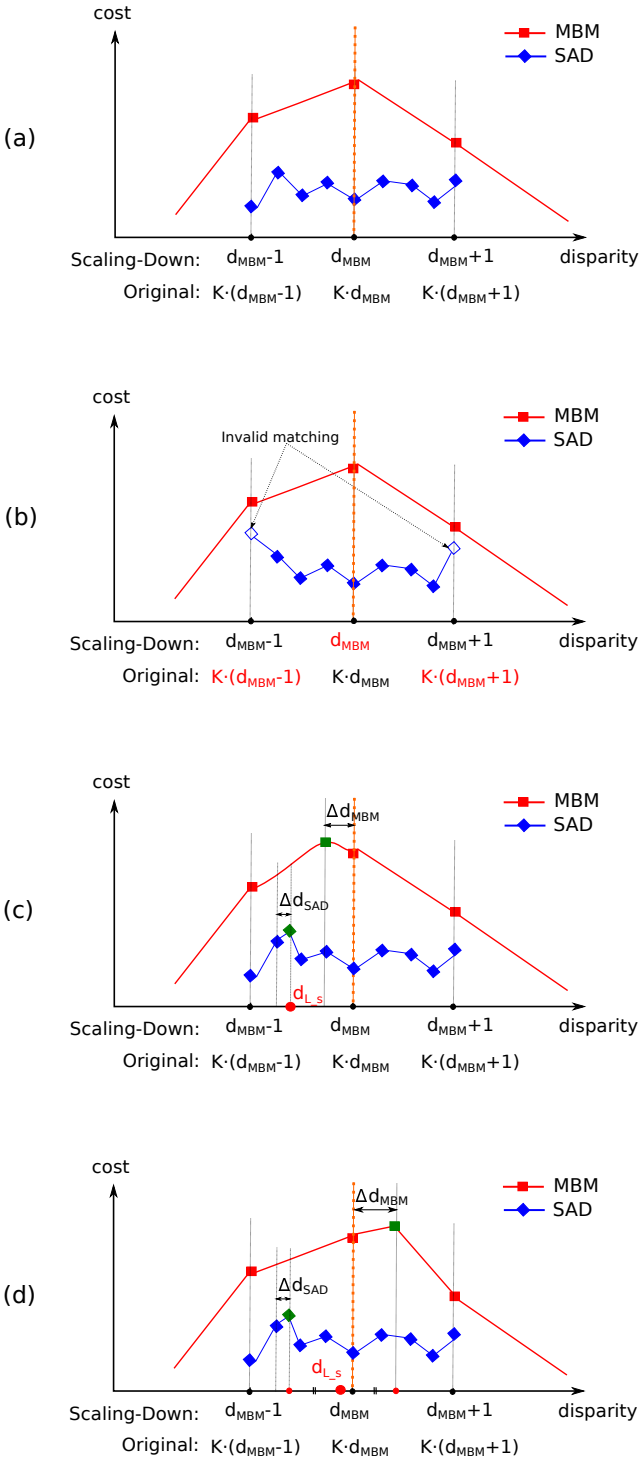

Figure 4.3: Fine-Tune. (a) Normal Matching. (b) Invalid Matching: The result of SAD is inconsistent with MBM. (c) Valid Matching: The results on the same side. (d) Valid Matching: The results on different sides.

1. If Δd_{MBM} and $d_{SAD} + \Delta d_{SAD} - K \cdot d_{MBM}$ have the same sign as shown in Fig.4.3(c), which means that both disparities are in the same side of the center line (red dotted line in Fig.4.3(c),(d)), the new disparity d_{L_S} for pixel $L(x, y)$ is calculated as $(d_{SAD} + \Delta d_{SAD})/K$.

2. If Δd_{MBM} and $d_{SAD} + \Delta d_{SAD} - K \cdot d_{MBM}$ have the different signs as shown in Fig.4.3(d), which means that the two disparities are in different side of the center line, the new disparity d_{L_S} for pixel $L(x, y)$ is calculated as the average of them $(d_{MBM} + \Delta d_{MBM} + (d_{SAD} + \Delta d_{SAD})/K)/2$.

During this matching step, although the matchings are performed twice, as the disparity range is limited in the secondary matching, only $D_{max}/K + 2K + 1$ matches are required for each pixel. Hence, not only an accurate matching can be ensured, but also the amount of the computation is kept small.

4.2 Implementation on GPU

We implement our algorithm on both NVIDIA GTX780 Ti and GTX1080 Ti GPUs, which have the different architectures. Although the number of streaming multi-processors (SMs) and the memory sizes are different, the hierarchy and the attribute of the memories have not any changed.

4.2.1 System Pipeline

Fig.4.4 shows the pipeline of our stereo vision system. The original color images are first converted to gray-scale on CPU, and then they are transferred to the global memory of GPU. The data size can be reduced to 1/3 (24bit to 8bit), and their transfer time also can be reduced. In our system, taking the GTX1080 Ti as an example, for the 2888×1920 pixels images, the processing on CPU takes about 4.81 milliseconds, and the processing on GPU takes about 13.7 milliseconds. Although the delay for the first frame takes more than 18 milliseconds, as shown in Fig.4.4, the calculation on the two devices can be run in parallel and the stream processing makes it possible to achieve the real-time processing as its throughput.

Since the main processing of our system is on the GPU side, we focus on how to implement our system on GPU.

4.2.2 Task Assignment and Data Mapping on GPU

Because of the high locality of the MBM algorithm, there exist many alternatives for how to process the pixels in parallel by using many cores on GPU. In our implementation, as shown in Fig.4.5, the image is divided by N along the y-axis. Here, Y is the height of the image and N is the number of the SMs of the

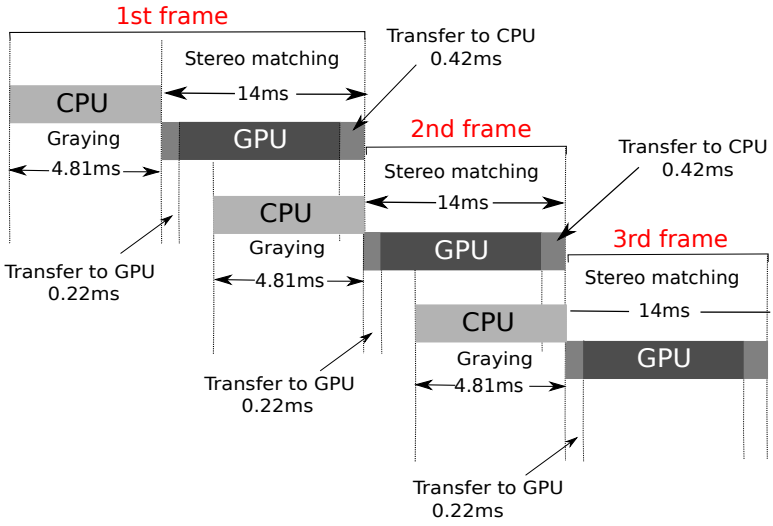

Figure 4.4: System Pipeline

target GPU. As shown in Fig.4.5, Y/N lines are assigned to each SM. In each SM, Y/N lines are processed from the top to bottom line by line. In this line by line processing, one pixel is assigned to one thread as shown in Fig.4.5-top and the pixels on the same line are processed in parallel. In our implementation, the whole work-flow is divided into five steps as shown in Fig.4.5, and in each step, the outputs of the previous step are fetched from the global memory into the on-chip memory, and the outputs of the current step are sent to the global memory for the next step. By reducing the number of steps, higher processing speed is expected because the number of the memory accesses to the global memory can be reduced. However, the size of our target image is large, and only the data required for processing one line can be held on on-chip shared memory and registers. Under this limitation, the five steps are the minimum set. In these five steps, the procedures that can be performed by using the data of the same line are packed in the same step such as the calculation of NCC cost and their aggregation along the x-axis in the step 2.

4.2.3 Effective Matching Processing on GPU

The most time consuming steps in our algorithm are step 2 and step 3, because in these steps, D matching costs are calculated. In order to achieve high performance, the optimization of these two steps is highly important. The following describes the details of our methods to improve the performance of these two steps.

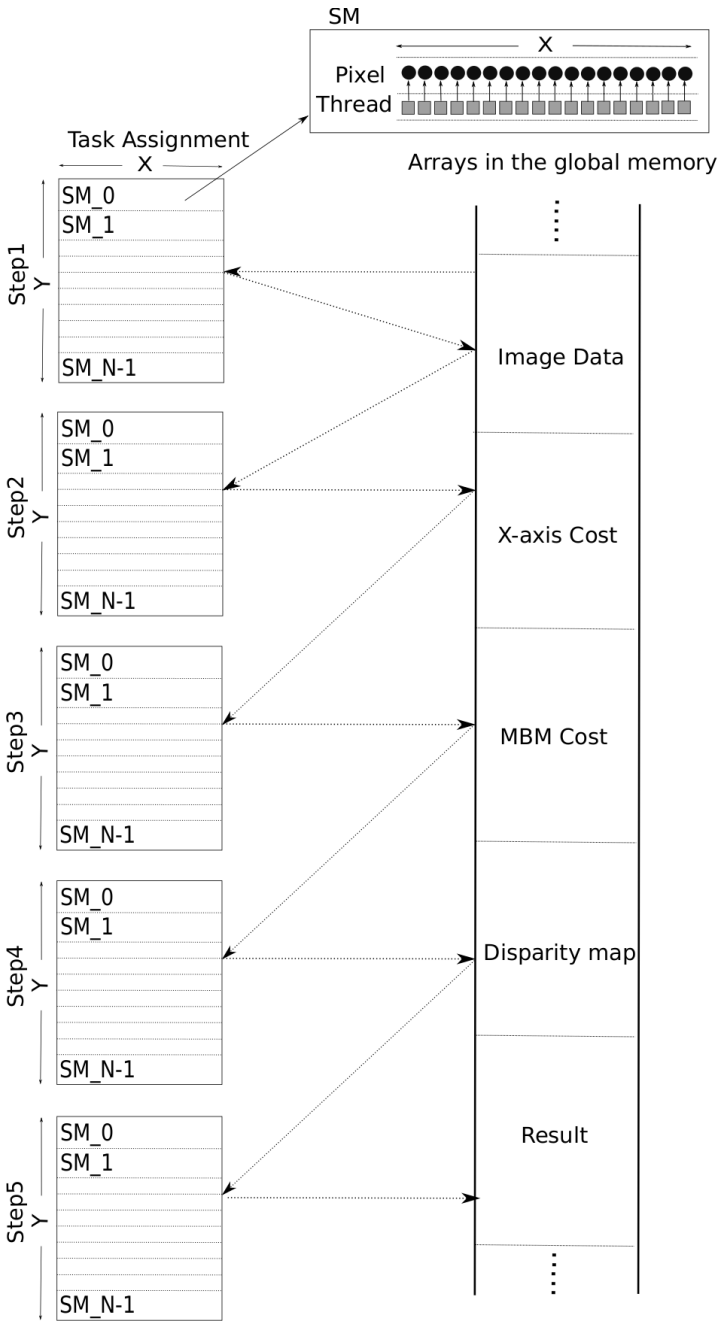

Figure 4.5: Task assignment to each step on GPU. *Step1:* Smoothing & Scaling Down. *Step2:* NCC calculation & Cost Aggregation along the x-axis. *Step3:* Cost Aggregation along the y-axis. *Step4:* WTA, Secondary Matching & Scaling Up. *Step5:* Cross-Check & Improvement.

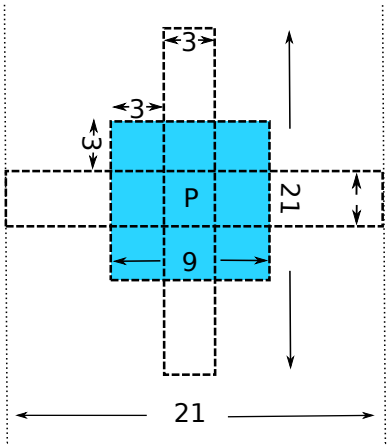

Figure 4.6: Multi-Block Matching

Making Full Use of On-chip Memory

Each SM in GPU has on-chip shared memory and registers in it. By using them efficiently, the access to global memory can be reduced. The data in the shared memory can be accessed from any threads in the same SM, but the data in the registers can be accessed only by the thread that wrote them in the registers. Table.4.1 shows the memory usage in the NCC calculation step. In this step, it is necessary to store 3×2 lines of the reference (left) and target (right) image in the shared memory. $\sigma_L(x, y)$, $\bar{L}(x, y)$, $\sigma_R(x - d, y)$ and $\bar{R}(x - d, y)$ are used for calculating NCC as described in Section 2.1.3, and each of them is accessed D times for calculating all NCCs. During the computations, $\sigma_L(x, y)$ and $\bar{L}(x, y)$ are accessed by only one thread, while $\sigma_R(x - d, y)$ and $\bar{R}(x - d, y)$ are accessed from D threads. Thus, $\sigma L(x, y)$ and $\bar{L}(x, y)$ can be stored in the on-chip registers. As shown in Table.4.1, the intermediate results for calculating NCCs (Costtemp and Result) are stored in the shared memory and the total size of the data stored in the shared memory reaches 42KB for calculating one line. This means that data for one line in the original size image cannot be stored in the shared memory, and reducing the image size is a must for the efficient computation by using only the on-chip memory.

In our implementation, the NCCs are calculated line by line. First, two sets of three lines $y-1$, y and $y+1$ are held in the shared memory, and the NCC costs of the center line y are calculated. Then, the pixels of the next line $y+2$ are fetched from the global memory, and the pixels of the oldest line $y-1$ are replaced by the new ones. Using the new line $y+2$ and the two lines that are already held on the chip y and $y+1$, the NCC costs of the next line $y+1$ are calculated.

Table 4.1: Memory Sharing in The Ncc Cost Calculation

Data Type	**Data Size**	
	Shared Memory	***Register***
Image Data	$768\times3\times4\times16(bit)=18(KB)$	
Ave(T)	$768\times2\times32(bit)=6(KB)$	
SD(T)	$768\times2\times32(bit)=6(KB)$	
Ave(R)		$768\times2\times32(bit)=6(KB)$
SD(R)		$768\times2\times32(bit)=6(KB)$
Costtemp	$768\times2\times32(bit)=6(KB)$	
Result	$768\times2\times32(bit)=6(KB)$	
Total	***42KB***	***12KB***

Image Data*: The image data with integer type which are generated in the step1 (Both left and right image).* ***Ave****: The average of the image data.* ***SD****: Standard deviation of the image data.* ***T****: Target image.* ***R****: Reference image. The "Ave" and the "SD" of the target image are stored in shared memory. On the other hand, those of the reference images are stored in the register.* ***Costtemp****: The result of the NCC.* ***Result****: the costs aggregated along the x-axis.*

Reusing the Intermediate Results

After the NCC cost calculation, they are aggregated using the MBM algorithm. As described in Section 4.2.2, the NCC costs of the next line cannot be calculated at the same time, which means that the NCC cost cannot be aggregated along the y-axis without using the global memory. Thus, two steps are required for the cost aggregation. In our MBM, three blocks with different size and shape are used. By choosing the block width properly, the intermediate results of the cost aggregation for the small blocks can be reused for larger blocks. Fig.4.6 shows a set of the blocks, the sizes of which are 3×21, 21×3, 9×9 respectively. With this combination, the partial sums calculated for the smallest block 3×3 can be reused for other blocks. Additionally, according to our evaluation, this combination also shows a good accuracy. By using larger blocks, higher matching accuracy can be expected, but according to our experience, the improvement is marginal though it requires more operations (addition) which may effect the processing speed.

Fig.4.7(a) shows the cost aggregation along the x-axis. For pixel $L(x, y)$, three cost aggregation steps are taken. First, the costs of itself $C(x, y, d)$ and its two neighbors, $C(x-1, y, d)$ and $C(x+1, y, d)$, are aggregated by the corresponding

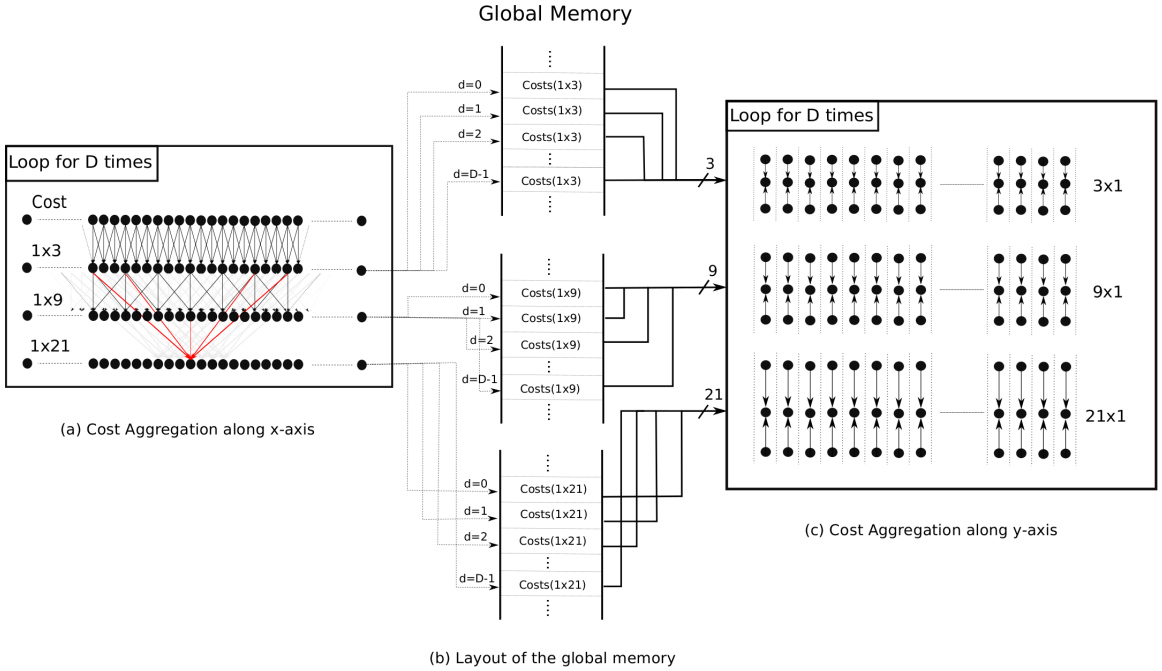

Figure 4.7: System Pipeline

thread. Then, its sum $C_3(x, y, d)$ is kept in the shared memory to be reused for other size blocks. Secondly, each thread aggregates $C_3(x, y, d)$, $C_3(x-3, y, d)$ and $C_3(x+3, y, d)$, and the $C_9(x, y, d)$ is obtained. $C_9(x, y, d)$ is stored into the register instead of the shared memory because it is not accessed from other threads in the following steps. Thirdly, $C_{21}(x, y, d)$ is calculated by adding $C_9(x, y, d)$ and its four neighbors, $C_3(x \pm 6, y, d)$ and $C_3(x \pm 9, y, d)$. All the costs are stored in the global memory to be used for the aggregation along the y-axis. This sequence is repeated D times ($d = [0, D)$) for each pixel.

Fig.4.7(b) shows the layout of the partial sums along the x-axis in the global memory. For each pixel, the cost of each disparity is stored in the global memory in the order that can make the next aggregation step run efficiently. Fig.4.7(c) shows the costs aggregation along the y-axis. Before entering this step, the cost aggregation of all lines along the x-axis is finished, and the all sums are stored in the global memory. For each pixel $L(x, y)$, its three aggregation costs are given as

follows:

$$C_{21\times3}(x,y,d) = \sum_{i=-1}^{1} C_{21}(x,y+i,d) \tag{4.3}$$

$$C_{9\times9}(x,y,d) = \sum_{i=-4}^{4} C_{9}(x,y+i,d) \tag{4.4}$$

$$C_{3\times21}(x,y,d) = \sum_{i=-10}^{10} C_{3}(x,y+i,d) \tag{4.5}$$

In these equations, C_{21}, C_9, C_3 are transferred from the global memory to the shared memory and are aggregated line by line. Here, suppose that $C_{21\times3}(x,y,d)$, $C_{9\times9}(x,y,d)$ and $C_{3\times21}(x,y,d)$ are held on the shared memory. Then, the three sums for the next line can be calculated efficiently as follows:

$$\begin{aligned} C_{21\times3}(x,y+1,d) &= \sum_{i=0}^{2} C_{21}(x,y+i,d) \\ &= C_{21\times3}(x,y,d) + C_{21}(x,y+2,d) - C_{21}(x,y-1,d) \end{aligned} \tag{4.6}$$

$$\begin{aligned} C_{9\times9}(x,y+1,d) &= \sum_{i=-3}^{5} C_{9}(x,y+i,d) \\ &= C_{9\times9}(x,y,d) + C_{9}(x,y+5,d) - C_{9}(x,y-4,d) \end{aligned} \tag{4.7}$$

$$\begin{aligned} C_{3\times21}(x,y+1,d) &= \sum_{i=-9}^{11} C_{3}(x,y+i,d) \\ &= C_{3\times21}(x,y,d) + C_{3}(x,y+11,d) - C_{3}(x,y-9,d) \end{aligned} \tag{4.8}$$

By this calculation method, the computation order of the aggregation along the y-axis can be reduced to O(1).

4.2.4 Subsequent processing on GPU

After MBM, a series of processing shown in Fig.4.5 are executed line by line. The task assignment to the threads is the same as the NCC cost calculation step, and all of the steps are executed one by one in each line:

1. The costs of MBM are transferred from the global memory to the shared memory repeatedly, and two initial disparity lines D'_{LMBM} and D'_{RMBM} are generated by the WTA. During the WTA, when the matching cost for $d_c + 1$, namely $C(x,y,d_c+1)$, is calculated, $C(x,y,d_c-1)$ and $C(x,y,d_c)$ are being held on the registers, and the sub-pixel estimation is performed for

d_c. With this implementation, when the integer disparity d_{MBM} which gives the maximum matching cost is obtained, an offset Δd_{MBM} is also obtained through the sub-pixel estimation.

2. Based on d_{MBM}, the secondary matching is executed in the range of $[K \cdot (d_{MBM} - 1), K \cdot (d_{MBM} + 1)]$. The same as the MBM, in the SAD matching, one thread corresponds to one pixel. For each pixel, d_{SAD} and Δd_{SAD} are calculated by using the same method as 1). Unlike the first matching, in the secondary matching, several lines of original image need to be transferred to the on-chip memory, and the amount of the data becomes usually several times the data that were used in the first matching. By using more data in this step, higher matching accuracy can be expected, but it requires more arithmetic operations and more memory space. Hence, according to the limitation of the hardware resources and the processing speed, choosing a suitable amount of data is very important. Here, one thing to note is that since the image data are also used during the Scaling-up step, the number of lines must be K at least.

3. The remaining steps are executed. During the Scaling up step, instead of storing the scaled up disparity map data successively (Fig.4.8(a)), the interpolated data are stored separately as shown in Fig.4.8(b). These data are reverted to the original order before the Cross-Check step. This method is used to avoid shifting the disparity of the scaled down images when they are fetched from the global memory.

4.3 Experimental Results

We have implemented the algorithm on a middle-end GPU NVIDIA GTX780 Ti and a high-end GPU NVIDIA GTX1080 Ti respectively, and evaluated the processing speed and the error rate using the Middlebury V3 [16], KITTI2012 [26] and KITTI2015 [27] benchmarks. In this section, we first evaluate the accuracy and processing speed of our system using each benchmark, and then make a comprehensive comparison with other systems.

4.3.1 Middlebury Benchmark

Fig.4.9 shows an accuracy comparison of our proposed method with the *Original-MBM* (MBM on the original image set) and the *Scaling_MBM* (MBM on the scaled down image set without secondary matching). In Fig.4.9, H and F are the image sizes. Images in F are larger than those in H. Their sizes are shown in Table.4.3. In this evaluation, $K = 2$ for the H-size images, and $K = 4$ for F-size

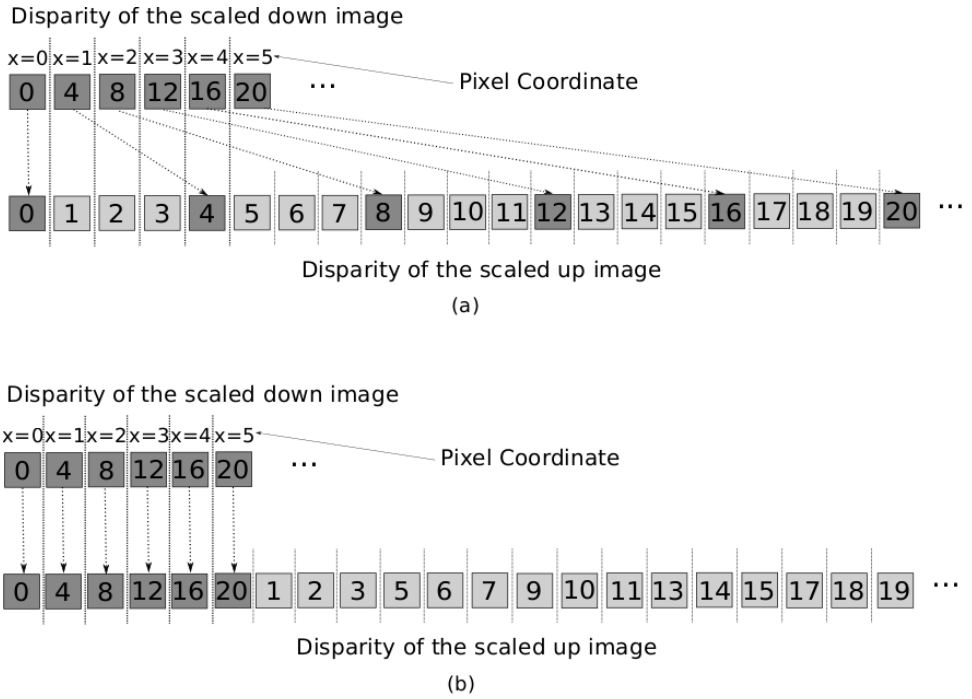

Figure 4.8: Effective Scaling-up (K=4)

images. For achieving higher processing speed, the block size of SAD is fixed to 3×3. T (a threshold introduced in Section 2.3.2) is set to K. In Fig.4.9, the x-axis shows several combinations of block sizes that are used in MBM. The sizes of all blocks are chosen so that the intermediate results of smaller blocks can be reused for larger blocks. The block sets shown in parentheses are used for the original size images and their sizes are two times those for the scaled down images. The y-axis shows the bad 2.0 error rate (percentage of "bad" pixels whose disparity are different more than 2.0) of the above three algorithms for the training image set. As shown in this graph, when the blocks are too large, their error rates become worse. For the *Scaling-MBM* and *Scaling-MBM+SAD*, the $3 \times 21, 21 \times 3, 9 \times 9$ block set shows the lowest error rate, and for the *Original-MBM*, the $5 \times 53, 53 \times 5, 17 \times 17$ shows the lowest error rate. *Scaling-MBM* shows the worst error rate (roughly 6% higher than other methods), and our method *Scaling-MBM+SAD* shows the lowest for both of H-size and F-size data sets. Furthermore, the error rates for both size images are almost the same, which shows that our methods is very robust. Here, it can be noted that the accuracy of our methods are always better than the *Original-MBM*, even though the information of original images is lost by down-scaling.

Fig.4.10 shows the results of four H-size images in the Middlebury Benchmark [16]. The block size used in *Scaling-MBM* and *Scaling-MBM+SAD* is $3 \times 21, 21 \times$

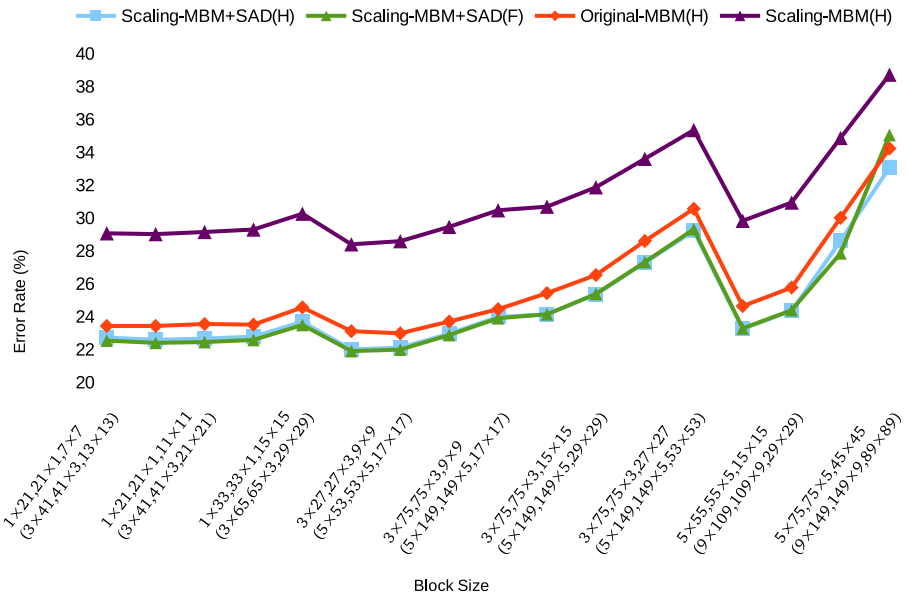

Figure 4.9: Accuracy Comparison. *Scaling-MBM+SAD*: Result of MBM on scaled down images with a secondary SAD matching. *Original-MBM*: Result of MBM on original images. *Scaling-MBM*: Result of MBM on scaled down images without secondary matching. *(H)*: H-size dataset. *(F)*: F-Size dataset.

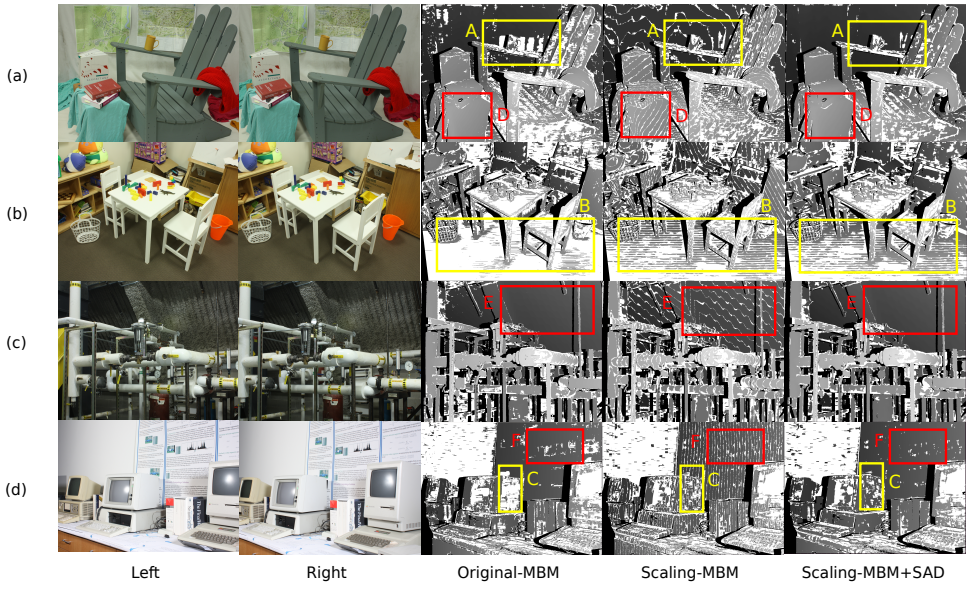

Figure 4.10: Matching Result. (a) Adirondack (b) Playtable (c) Pipes (d) Vintage. A: Repetitive patterns. B: Perspective distortions & Uniform regions. C: Uniform regions. D,E,F: Gradient regions.

$3, 9 \times 9$, and the block size of SAD is 3×3, while the block size used in *Original-MBM* is $5 \times 41, 41 \times 5, 17 \times 17$. Three areas, A, B and C, show the repetitive patterns, perspective distortions and uniform regions respectively, which represent the three kinds of difficult problems for the block matching method. D, E and F show the gradient regions for which it is required to decide their disparities considering their continuity. The white pixels represent the matching errors. As shown in this figure, our approach shows better results than other approaches specially in those marked areas. This means that in our approach, GCPs are correct as well as the *original-MBM*, and the disparities of non-GCPs are improved better. For A,B and C regions, because of the scaling down, some information that makes the matching difficult in the original size images, such as repetition of patterns and a serious of similar pixels, are discarded, and better matching becomes possible. Furthermore, because of the secondary matching, our method, Scaling-MBM+SAD, shows a better result than the Scaling-MBM in these regions. For D, E and F, a secondary matching generates the continuous disparities, and the disparities are improved to the same level as matching by *Original-MBM*.

Table.4.2 shows the accuracy comparison among several stereo vision systems. In this table, systems that were evaluated using Middlebury and KITTI benchmarks are listed by their average error rates on Middlebury benchmark. Our error rate for F-size is listed at the bottom, because Table.4.2 is a hard copy of the benchmark evaluation site [16], and in this site, it is not allowed to upload more than one result at a time. Our error rates for H-size and F-size image sets are not the top, but they are not bad compared with other systems.

Table.4.3 shows the details of the processing speed of our system on GTX1080 Ti. We select 8 representative image sets from the Middlebury Benchmark [16]. 4 sets of them are H-size, and 4 sets are F-size. In the evaluated images, for example, Adirondack and MotorcycleE are large images with small disparity, Vintage is a large image with large disparity, and Teddy is a small image with small disparity. For each image set, the execution time of each processing step are shown. The graying step ($Gray$) which takes the large portion of the execution time, is executed on CPU. $HtoD$ and $DtoH$ steps show the data transmission time between the CPU and GPU. $DtoH$, the transmission time from GPU to CPU, is roughly twice as $HtoD$, CPU to GPU, because $HtoD$ is for two unsigned char images and $DtoH$ is for one float image. The transmission time of F-size data set is roughly 4 times that of H-size, because F-size images are 4 times larger than H-size images. Among the steps executed on GPU, the cost calculation and its aggregation ($NCC\&AggH$ and $AggV$) take most of the computation time, because frequent data transfer (D times loops) between the on-chip memory and off-chip memory is required. The execution time of the secondary matching step ($WTA\&SeM$) is relatively small, although several loops of matching are still needed. This is because SAD does not require the global memory access to save/read the intermediate results owing to

Table 4.2: Accuracy Comparison on Middlebury Benchmark

Date	Name	Res	Weight Avg	Adiron MP: 5.7 nd: 290 im0 im1 GT nonocc	ArtL MP: 1.5 nd: 256 im0 im1 GT nonocc	Jadepl MP: 5.2 nd: 640 im0 im1 GT nonocc	Motor MP: 5.9 nd: 280 im0 im1 GT nonocc	MotorE MP: 5.9 nd: 280 im0 im1 GT nonocc	Piano MP: 5.4 nd: 260 im0 im1 GT nonocc	PianoL MP: 5.4 nd: 260 im0 im1 GT nonocc	Pipes MP: 5.7 nd: 300 im0 im1 GT nonocc	Playrm MP: 5.3 nd: 330 im0 im1 GT nonocc	Playt MP: 5 nd: 290 im0 im1 GT nonocc	PlaytP MP: 5 nd: 290 im0 im1 GT nonocc	Recyc MP: 5.6 nd: 260 im0 im1 GT nonocc	Shelvs MP: 5.9 nd: 240 im0 im1 GT nonocc	Teddy MP: 2.7 nd: 256 im0 im1 GT nonocc	Vintge MP: 5.5 nd: 760 im0 im1 GT nonocc
05/31/18	iResNet_ROB	H	**2.50** 1	1.17 12	2.80 5	8.93 5	2.16 9	2.16 9	1.59 3	2.12 5	4.05 6	2.63 4	1.44 2	1.17 2	1.04 4	1.81 5	1.22 5	1.97 6
01/24/17	3DMST	H	4.59 2	1.16 10	5.25 25	23.9 19	2.99 16	2.94 13	2.01 13	3.68 17	6.17 17	3.53 9	1.68 6	1.62 8	1.30 8	4.77 18	1.83 21	2.82 13
04/19/15	MeshStereo	H	7.58 3	2.39 32	6.41 35	36.4 60	5.40 50	5.71 53	3.25 29	5.45 28	11.6 53	6.34 32	4.92 29	2.73 25	2.25 27	11.1 72	1.85 23	5.62 21
04/08/15	REAF	H	8.49 4	4.07 56	7.29 43	32.4 46	4.76 38	4.70 38	5.00 54	11.7 65	10.9 42	8.61 55	15.1 62	4.04 42	2.54 33	9.72 62	3.30 50	9.29 42
07/28/14	SGM	F	8.53 5	4.90 67	4.65 17	30.3 39	4.70 35	4.32 32	3.77 39	5.76 29	10.4 36	7.04 39	25.4 82	4.27 49	4.36 68	9.34 60	2.33 30	17.7 77
05/10/18	MSMD_ROB	F	9.22 6	2.85 40	8.58 57	45.1 77	5.12 43	4.99 42	3.75 38	7.65 42	11.0 44	6.86 37	9.76 48	9.32 77	2.74 37	3.58 14	3.02 47	9.59 44
01/15/17	IGF	Q	9.49 7	4.56 61	7.33 44	28.8 36	5.87 57	5.91 54	6.36 74	12.1 67	11.5 52	7.16 41	27.3 87	8.64 73	3.85 62	9.19 55	3.30 50	9.12 38
04/24/16	HLSC_cor	H	9.61 8	3.35 50	9.70 63	35.0 56	6.85 70	6.87 65	3.92 41	7.30 40	13.8 73	10.1 65	16.6 67	3.90 40	3.55 56	11.7 77	2.99 45	14.6 61
09/18/14	SNCC	H	10.4 9	3.63 54	6.74 37	39.8 66	5.12 43	5.11 44	4.65 49	8.23 46	11.8 56	8.05 50	45.6 96	4.36 54	3.29 51	8.10 40	2.52 32	14.8 62
03/26/18	ELAS_ROB	H	10.5 10	4.08 57	7.18 42	52.8 81	5.40 50	5.47 47	5.00 54	9.14 52	10.7 38	8.03 49	23.3 79	3.83 38	3.79 60	9.40 61	3.27 49	10.5 48
06/08/18	MBM	H	10.7 11	4.31 60	9.25 60	37.0 61	5.84 56	5.65 51	7.10 76	13.9 71	11.5 51	12.6 70	24.4 81	7.68 69	3.49 54	12.5 81	3.90 60	13.3 57
10/13/15	MDP	H	10.8 12	1.56 16	7.37 45	53.8 82	5.89 58	6.18 61	4.04 43	8.81 50	14.2 75	11.0 67	15.8 66	4.19 45	4.00 65	9.24 56	3.95 62	15.2 64
11/13/17	CBMV	H	11.5 13	5.98 76	11.0 67	41.2 70	5.26 47	5.51 48	4.03 42	14.1 72	11.2 46	11.5 68	8.45 39	6.89 66	3.84 61	8.45 44	7.11 73	40.2 90
08/28/15	MC-CNN-acrt	H	11.8 14	4.24 59	18.7 86	34.1 53	7.21 72	7.22 68	6.00 68	9.35 53	13.5 68	18.3 76	9.71 47	9.37 78	4.64 70	6.62 29	9.31 80	21.6 84
11/24/16	ADSM	Q	12.3 15	14.3 90	10.6 66	34.1 52	6.00 62	8.00 72	7.37 77	20.4 82	12.1 60	16.9 72	25.5 83	5.84 61	5.83 81	17.2 88	4.11 63	11.1 53
05/01/18	PSMNet_ROB	Q	13.3 16	8.83 84	13.9 70	68.4 91	8.26 76	9.16 79	5.89 67	10.5 59	14.4 76	9.38 59	5.54 32	5.52 60	4.98 72	11.6 76	3.87 59	9.66 45
11/15/16	MC-CNN-WS	H	13.7 17	5.73 73	20.5 89	36.3 59	9.39 82	9.37 80	8.13 80	16.1 77	16.7 85	18.7 78	11.5 51	10.1 84	5.05 75	9.83 63	11.0 88	20.8 83
09/10/14	LAMC_DSM	H	14.6 18	7.65 81	21.8 91	37.9 63	11.3 87	11.1 82	8.81 82	11.7 66	17.4 87	22.7 84	15.4 63	10.6 85	5.91 82	13.4 83	10.2 82	15.5 65
08/31/14	BSM	Q	23.5 19	12.7 87	28.7 95	58.7 87	14.8 93	14.7 86	16.0 93	35.8 94	24.5 91	29.4 91	31.0 90	20.2 95	12.1 91	19.2 93	14.3 95	39.3 89
06/09/18	MBM	F	10.8 11	4.40 60	9.22 60	36.6 61	5.76 53	5.58 50	7.10 76	14.4 72	11.5 51	13.3 70	25.4 81	7.52 69	3.58 58	13.1 81	3.87 59	13.8 58

Size(MBM): $3 \times 21, 21 \times 3, 9 \times 9$. Size(SAD): 3×3. iResNet_ROB [29], 3DMST [41], MeshStereo [37], REAF [39], SGM [19], MSMD_ROB [36], IGF [40], HLSC_cor [31], SNCC [58], ELAS_ROB [21], MDP [35], CBMV [34], MC-CNN-arct [5], ADSM [32], PSMNet_ROB [38], MC-CNN-WS [18], LAMC_DSM [30], BSM [33]

Table 4.3: Execution Time With the Middlebury Benchmark Set (ms)

Image	Size	Dmax	Gray(CPU)	HtoD	SS	NCC&AggH	AggV	WTA&SeM	CC	DtoH	Overall(GPU)
Adirondack(H)	1436×992	145	5.31	0.226	0.37	1.135	2.68	0.684	0.274	0.432	5.801
MotorcycleE(H)	1482×994	128	4.98	0.233	0.039	1.073	2.652	0.678	0.254	0.447	5.366
Teddy(H)	900×750	128	1.9	0.11	0.029	0.51	0.875	0.309	0.205	0.205	2.243
Vintage(H)	1444×960	380	4.81	0.219	0.038	3.096	7.6	1.779	0.545	0.422	13.699
Adirondack(F)	2872×1984	290	23.88	0.921	0.052	1.191	2.693	1.029	1.334	1.764	8.984
MotorcycleE(F)	2964×1988	256	23.02	0.931	0.054	1.118	2.63	1.007	1.162	1.788	8.69
Teddy(F)	1800×1500	256	10.29	0.428	0.037	0.537	1.271	0.539	0.82	0.82	4.452
Vintage(F)	2888×1920	760	21.43	0.889	0.053	3.248	8.189	2.288	2.322	1.699	18.688

Size(MBM): 3×21, 21×3, 9×9 ***Size(SAD)***: 3×3 ***Dmax***: *Maximum Disparity* ***Gray***: *Graying.* ***HtoD***: *Data transmission time from CPU to GPU.* ***SS***: *Smoothing & Scaling-Down.* ***NCC&AggH***: *NCC cost calculation & Aggregation along the x axis.* ***AggV***: *Aggregation along the y axis.* ***WTA&SeM***: *WTA & Secondary Matching.* ***CC***: *Cross_Check.* ***DtoH***: *Data transmission time from GPU to CPU.* **Overall**: The overall time taken on GPU.

the small amount data required in this step. The overall time required on GPU is proportional to the image size and the maximum disparity. As shown in *overall* field, even for the largest image set "Vintage", its processing time is only 18.688ms, which makes our real-time processing possible. However, in the processing for F-size data set, it can be noted that the time required by CPU is longer than GPU, and the processing speed depends on the graying step on CPU.

Fig.4.11 compares the processing speed when the block sizes are changed in our system. As shown in this figure, in all cases, more computation time is required for larger blocks. Considering the error rate shown in Fig.4.9, the block sizes used in our implementation, $3 \times 21, 21 \times 3, 9 \times 9$, gives the good balance of the processing speed and the matching accuracy. Table.4.4 shows the comparison of our processing speed with other systems listed in Table.4.2. Here, because the image sizes and the disparity ranges are different, in order to make a clear comparison, we chose the systems that can process the H-size image sets, and used the *Geometric Mean* method [28] to compare them. Due to the space limitation, Table.4.4 is folded into two (each system on the leftmost column is evaluated using 15 images (8 in the upper part, and 7 in the lower part)). In this table, for each system, two values are shown for each corresponding image. The first one is its runtime, and the second one in the parentheses is the normalized value by our system (GTX 780Ti). From these values, one *Geometric Mean* value can be calculated for each system. Based on this value, the difference of the processing speed among these systems can be shown obviously. As shown in the column *Geometric Mean* of Table.4.4, the processing speed of our system is much faster than other system. The processing speed by GTX 1080Ti is three times faster than GTX 780Ti.

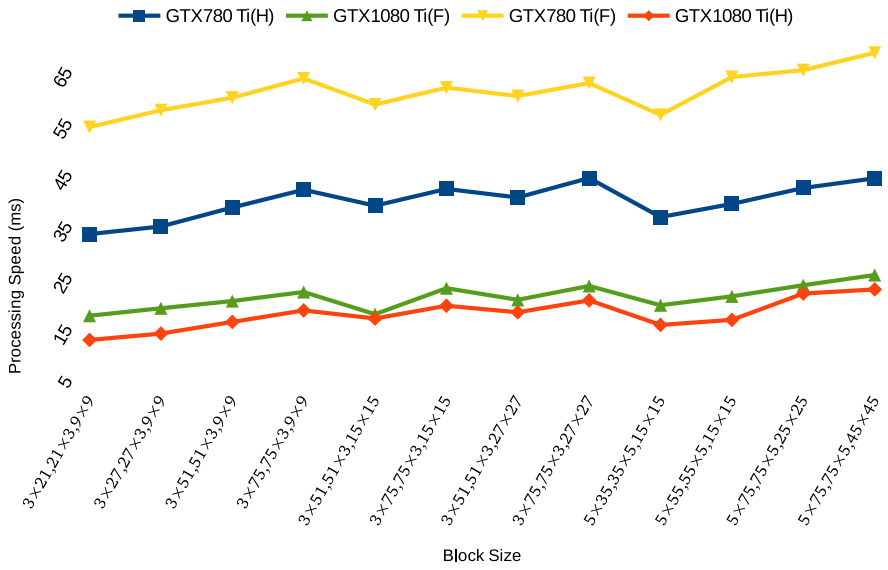

Figure 4.11: Processing Speed Comparison.

Table 4.4: Processing Speed Comparison (sec)

System	Adirondack	ArtL	Jadeplant	Motorcycle	MotorcycleE	Piano	PianoL	Pipes
IresNet [29]	0.35 (23.33)	0.28 (70)	0.35 (12.96)	0.35 (23.33)	0.35 (23.33)	0.34 (24.29)	0.34 (24.29)	0.34 (21.25)
ELAS [21]	0.52 (34.67)	0.13 (32.5)	0.49 (18.15)	0.54 (36)	0.55 (36.67)	0.5 (35.71)	0.49 (35)	0.49 (30.63)
MSMD [36]	0.9 (60)	0.2 (50)	0.8 (29.63)	0.9 (60)	0.9 (60)	0.8 (57.14)	0.8 (57.14)	0.9 (56.25)
SNCC [20]	1.03 (68.67)	0.24 (60)	1.67 (61.85)	1.05 (70)	1.09 (72.67)	0.82 (58.57)	0.82 (58.57)	1.07 (66.88)
MC-CNN-WS [18]	2.65 (176.67)	0.63 (157.5)	4.14 (153.33)	2.58 (172)	2.58 (172)	2.25 (160.71)	2.25 (160.71)	2.59 (161.88)
REAF [39]	4.37 (291.33)	0.78 (195)	9.73 (360.37)	4.43 (295.33)	4.43 (295.33)	3.32 (237.14)	3.31 (236.43)	4.7 (293.75)
MDP [35]	55.5 (3700)	12.8 (3200)	82.2 (3044)	71.8 (4787)	71.6 (4773)	60 (4286)	64.1 (4578)	78.3 (4894)
Meshstereo [37]	59.1 (3940)	49.8 (12450)	87 (3222)	63 (4200)	63 (4200)	62.2 (4443)	62.5 (4464)	66.6 (4163)
MC-CNN-acrt [5]	100 (6666.67)	23.9 (5975)	206 (7630)	109 (7267)	108 (7200)	99 (7071)	96.5 (6893)	108 (6750)
3DMST [41]	178 (11867)	39 (9750)	203 (7519)	203 (13533)	199 (13267)	187 (13357)	184 (13143)	171 (10688)
LAMC-DSM [30]	519 (34600)	158 (39500)	1034 (38296)	493 (32867)	495 (33000)	477 (34071)	542 (38714)	573 (35813)
HLSC [31]	1285 (85667)	147 (36750)	2281 (84481)	1933 (128867)	1899 (126600)	1340 (95714)	1279 (91357)	1689 (105563)
CBMV [34]	424 (28266.67)	245 (61250)	1207 (44703)	605 (40333)	618 (41200)	520 (37143)	526 (37571)	609 (38063)
our system(1080Ti)	0.005 (0.33)	0.001 (0.25)	0.01 (0.37)	0.005 (0.33)	0.005 (0.33)	0.004 (0.29)	0.005 (0.36)	0.005 (0.31)
our system(780Ti)	0.015 (1)	0.004 (1)	0.027 (1)	0.015 (1)	0.015 (1)	0.014 (1)	0.014 (1)	0.016 (1)

System	Playroom	Playtable	PlaytableP	Recycle	Shelves	Teddy	Vintage	Geometric Mean
IresNet [29]	0.35 (21.88)	0.34 (24.29)	0.35 (25)	0.35 (26.92)	0.35 (25)	0.32 (45.71)	0.35 (10.29)	**20.8**
ELAS [21]	0.5 (31.25)	0.49 (35)	0.46 (32.86)	0.5 (38.46)	0.54 (38.57)	0.23 (32.86)	0.51 (15)	**26.13**
MSMD [36]	0.8 (50)	0.7 (50)	0.7 (50)	0.8 (61.54)	0.9 (64.29)	0.4 (57.14)	0.8 (23.53)	**41.18**
SNCC [20]	1.08 (67.5)	0.9 (64.29)	0.91 (65)	0.95 (73.08)	0.94 (67.14)	0.42 (60)	2.06 (60.59)	**49.3**
MC-CNN-WS [18]	2.67 (166.88)	2.21 (157.86)	2.24 (160)	2.22 (170.77)	2.31 (165)	1.06 (151.43)	4.84 (142.35)	**116.19**
REAF [39]	4.75 (296.88)	4.07 (290.71)	4.17 (297.86)	3.69 (283.85)	3.78 (270)	1.32 (188.57)	12.4 (364.71)	**185.88**
MDP [35]	62.7 (3919)	48.8 (3486)	55.5 (3964)	55.4 (4262)	79.8 (5700)	26.1 (3729)	28.6 (841)	**2358**
Meshstereo [37]	66.2 (4138)	58.4 (4171)	57.9 (4136)	56.9 (4377)	59.5 (4250)	50.4 (7200)	71.5 (2103)	**2648**
MC-CNN-acrt [5]	122 (7625)	104 (7429)	101 (7214)	92.6 (7123)	94.5 (6750)	40.5 (5786)	264 (7765)	**3846**
3DMST [41]	171 (10688)	171 (12214)	189 (13500)	177 (13615)	178 (12714)	87 (12429)	208 (6118)	**6354**
LAMC-DSM [30]	567 (35438)	486 (34714)	482 (34429)	481 (37000)	578 (41286)	217 (31000)	961 (28265)	**17730**
HLSC [31]	1514 (94625)	1146 (81857)	1177 (84071)	1112 (85538)	1470 (105000)	378 (54000)	1905 (56029)	**40440**
CBMV [34]	602 (37625)	559 (39929)	565 (40357)	9999 (769154)	493 (35214)	382 (54571)	1464 (43059)	**24310**
our system(1080Ti)	0.005 (0.31)	0.004 (0.29)	0.005 (0.36)	0.004 (0.31)	0.004 (0.29)	0.002 (0.29)	0.013 (0.38)	**0.34**
our system(780Ti)	0.016 (1)	0.014 (1)	0.014 (1)	0.013 (1)	0.014 (1)	0.007 (1)	0.034 (1)	**1**

Geometric Mean: The Geometric Mean is calculated from “Adirondack” to “Vintage” image sets.

Table 4.5: KITTI2012

Error	Out-Noc	Out-All	Avg-Noc	Avg-All
2 pixels	7.86 %	9.53 %	1.2 px	1.4 px
3 pixels	5.45 %	6.88 %	1.2 px	1.4 px
4 pixels	4.38 %	5.56 %	1.2 px	1.4 px
5 pixels	3.70 %	4.67 %	1.2 px	1.4 px

Table 4.6: KITTI2015

Error	D1-bg	D1-fg	D1-all
All / All	6.06 %	14.13 %	7.40 %
All / Est	6.06 %	14.13 %	7.40 %
Noc / All	5.41 %	12.72 %	6.62 %
Noc / Est	5.41 %	12.72 %	6.62 %

4.3.2 KITTI Benchmark

We also evaluated our system using KITTI2012 [26] and KITTI2015 [27] benchmarks, separately. In these benchmarks, hundreds of images are divided into two groups. For the first group ("training dataset"), their true disparity maps are given, and this group is used to tune the parameters of the stereo vision systems. The users are required to upload their disparity maps for the second group ("testing dataset") to the website, and their matching accuracy are evaluated on the website. The image sizes are close to 1250×375 and the ranges of disparities are always 256. Here, since the numbers of lines in KITTI are small, we only scaled down the images along the x-axis (K=2). According to our evaluation, the runtime of our system on GTX 780Ti is roughly 0.015s (66.7 fps), and it is 0.005s (200 fps) on GTX 1080Ti. For the "testing dataset" (the second group), our accuracy is shown in Table.4.5 and Table.4.6, which was ranked 73rd out of 108 systems in KITTI2012 and ranked 94th out of 119 systems in KITTI2015. This accuracy is not good, but for this evaluation, the same parameters as *Middlebury* are used, and they are not tuned for KITTI benchmarks. As for the accuracy of the "training dataset" (for this first group, we can know the error rate of each image), we achieved the lowest error rate 0.084% for "KITTI2012-000000" and 0.058% for "KITTI2015-000135", while the worst error rate 20.6% for "KITTI2012-000180" and 67.8% for "KITTI2015-000104" as shown in Fig.4.12. According to our results, except the "KITTI2015-000104" that was taken in a tunnel, most of the error rates are kept between 1% and 5% on both benchmarks. We think that this error rate is enough for most practical use.

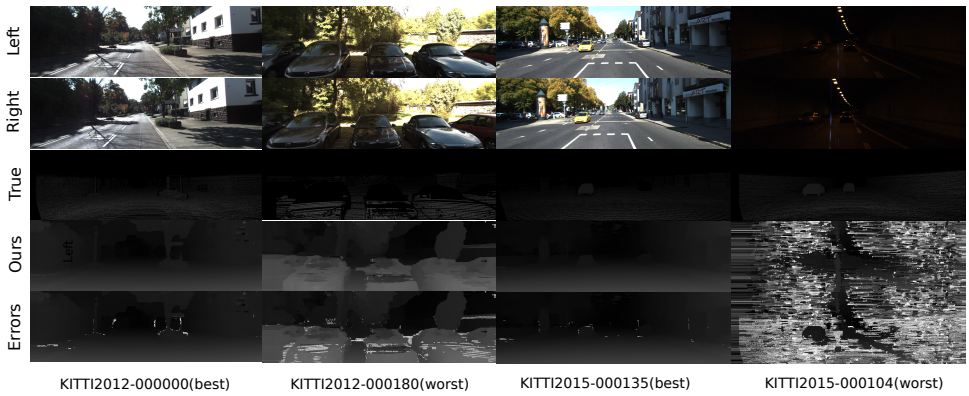

Figure 4.12: Evaluation results using the KITTI benchmarks.

Table 4.7: Accuracy Comparison

System	KITTI 2012 (%)	KITTI 2015(%)	Middlebury (%)	GM2012	GM 2015	GM Overall
Our system(H)	6.88 (1)	7.4 (1)	10.7 (1)	1	1	1
Our system(F)	6.88 (1)	7.4 (1)	10.8 (1.01)	1	1	1
MC-CNN-Acrt [5]	3.63 (0.53)	3.89 (0.53)	11.8 (1.1)	0.76	0.76	0.68
CBMV [34]	4.73 (0.69)	5.06 (0.68)	11.5 (1.07)	0.86	0.85	0.79
SGM [19]	7 (1.02)	6.38 (0.86)	8.51 (0.8)	0.9	0.83	0.89
PSMNET [38]	1.89 (0.27)	2.31 (0.31)	13.3 (1.24)	0.58	0.62	0.47
SNCC [20]	6.44 (0.94)	7.14 (0.96)	10.4 (0.97)	0.95	0.96	0.96
ELAS [21]	9.66 (1.4)	9.72 (1.31)	10.5 (0.98)	1.17	1.13	1.22
3DMST [41]	-	4.97 (0.67)	4.59 (0.43)	-	0.54	-
MDP [35]	-	5.36 (0.72)	10.8 (1.01)	-	0.85	-
MSMD [36]	-	4.16 (0.56)	9.22 (0.86)	-	0.69	-
Meshstereo [37]	-	8.38 (1.13)	7.58 (0.71)	-	0.9	-
REAF [39]	-	10.11 (1.37)	8.49 (0.79)	-	1.04	-
IGF [40]	-	10.84 (1.46)	9.49 (0.89)	-	1.14	-
IresNet [29]	2.16 (0.31)	-	2.5 (0.23)	0.27	-	-
MC-CNN-WS [18]	4.45 (0.65)	-	13.7 (1.28)	0.91	-	-
LAMC-DSM [30]	11.49 (1.67)	-	14.6 (1.36)	1.51	-	-
HLSC [31]	12.82 (1.86)	-	9.61 (0.9)	1.29	-	-
ADSM [32]	10.05 (1.46)	-	12.3 (1.15)	1.3	-	-
BSM [33]	13.44 (1.95)	-	23.5 (2.2)	2.07	-	-

Our system(H): The sizes of Middlebury image sets are H-size. **Our system(F)**: The sizes of Middlebury image sets are F-size. **GM 2012**: The Geometric Mean of the KITTI 2012 and Middlebury benchmarks. **GM 2015**: The Geometric Mean of the KITTI 2015 and Middlebury benchmarks. **GM Overall**: The Geometric Mean of the KITTI 2012, KITTI 2015 and Middlebury benchmarks.

4.3.3 Accuracy Comparison between Different Systems

In this subsection, we compare the error rates of all systems that are evaluated not only by using Middlebury, but also by using at least one of KITTI benchmarks. For this comparison, the *Geometric Mean* is used. As shown in Table.4.7, the 2nd, 3rd and 4th columns show the error rates (and the normalized one by our system) for Middlebury, KITTI 2012 and KITTI 2015 benchmark sets, and 5th, 6th and 7th columns show the Geometric Mean for Middlebury with KITTI 2012, KITTI 2015, and both. According to these results, it can be noted that the accuracy of our system in KITTI2012 and Middlebury is close to the medium level, and as described above, the accuracy of KITTI2015 is lower, which leads to a decrease in overall performance.

4.3.4 Speed Comparison between Different Systems

Table.4.8 compares the processing speed of stereo vision systems on different architectures. All the systems achieved a real-time processing as shown in *FPS* field, but their target image size (*Size*) and disparity range (*Dmax*) are different. *MDE/s* means mega disparity evaluation per second, and shows the true processing speed of each system. As shown in this table, *MDE/s* of our system is much higher than other systems. For calculating a disparity map of large size image such as 2888×1920, larger *Dmax* (760) is required, and with GTX 780Ti, its real-time processing cannot be achieved. However, by using faster GPU, GTX 1080 Ti, it becomes possible by our approach, and its *MDE/s* is 12x to 1060x faster than other systems. To compare the performance on different architectures, another criterion, *disparities/cycle*, is calculated by using equation (4.9), which means the number of disparities that can be processed per clock cycle on each architecture.

$$Disparities/cycle = \frac{ImageSize \times D_{max} \times FPS}{Frequency} \tag{4.9}$$

For GPU systems, the *frequency* refers to the frequency of each core, and for other systems, it means the frequency of the overall system. By comparing *disparities/cycle*, we can understand the performance gain by the algorithms implemented on each device. As shown in Table.4.8, for some systems, although their *MDE/s* are similar, there exists a gap in *disparities/cycle* such as "FPGA-Road [7]" and "FPGA-SOC [44]", because their frequency is different. Our rate 116.743 (for F-size images on GTX 1080Ti) is much faster than other GPU systems (even 18.04 for H-size images on GTX-780 Ti is faster). This means that our algorithm works very well on GPUs. However, FPGAs shows higher rate than our system. This comes from the fact that higher parallelism is possible on FPGAs than GPUs because the data width required in the stereo vision systems is less than 2B in

Table 4.8: Comparison With High-Speed Stereo Vision Systems

System	Size	Dmax	FPS	MDE/s	Hardware	Frequency	Disparities/Cycle	#Cores	Disparities/Cycle/Core
RT-FPGA [1]	1920×1680	60	30	5806	Kintex 7	160Mhz	36.288	-	-
FUZZY [2]	1280×1024	15	76	1494	Cyclone II	100Mhz	14.942	-	-
FPGA-Road [7]	1920×1080	128	50	13271	Virtex-6	110Mhz	120.645	-	-
Low-Power [8]	1024×768	64	30	1510	Virtex-7	50Mhz	30.198	-	-
FlexibleSV [42]	2048×1024	255	22	11765	Zynq 7045	200Mhz	58.82	-	-
FPGART [43]	1600×1220	128	42	10321	Stratix-IV	180Mhz	57.344	-	-
FPGA-SOC [44]	1280×720	256	60	14156	Zynq XC7Z030	200Mhz	70.77	-	-
RT-MULTI [45]	640×480	64	325	6390	Virtex-6	100Mhz	63.89	-	-
H-GF [48]	1280×720	60	60	3539	Kintex-7	145Mhz	24.40	-	-
ScalSGM [52]	1280×720	128	30	3438	VC709	130Mhz	27.20	-	-
LowEng [53]	1024×768	128	127	12784	Cyclone-4	100Mhz	127.84	-	-
RT-GF [54]	1920×1080	48	80	7962	Cyclone-4	108.5Mhz	73.38	-	-
BP-Stereo [49]	450×375	60	33	334	I5-4570	3200Mhz	0.104	-	-
Blind-RT [50]	640×360	99	30	684	Zynq ZC702	150Mhz	4.56	-	-
ADAS [51]	1280×720	256	30	7078	ADAS+FPGA	250Mhz	28.31	-	-
RGB-SV [47]	450×375	60	23	233	Quad-Core	2660Mhz	0.0875	4	0.021886
Comp-GPU [55]	640×480	128	55	2163	GTX 280	604Mhz	3.581	240	0.014919
IDR [13]	450×375	42	23	163	GTX TITAN BlACK	889Mhz	0.18	2880	0.000064
BF-DP [46]	340×240	16	192	251	GTX 580	772Mhz	0.324	512	0.000634
ETE [3]	1242×375	256	29	3458	GTX TITAN X	1000Mhz	3.45	3072	0.001126
EmbeddedRT [11]	640×480	128	81	1652	Tegra X1	1000Mhz	1.651	256	0.006415
Our System 1	1444×960	380	30	15803	GTX 780 Ti	876Mhz	18.04	2880	0.006264
Our System 2	1444×960	380	72	37927	GTX 1080 Ti	1480Mhz	25.626	3854	0.006649
Our System 3	2888×1920	760	18	75855	GTX 780 Ti	876Mhz	86.592	2880	0.030067
Our System 4	2888×1920	760	41	172781	GTX 1080 Ti	1480Mhz	116.743	3854	0.030292

many cases. To compare the performance of the systems on CPU and GPU, *disparities/cycle/core* is also shown in Table.4.8. This comparison shows that in our system, each core works more efficiently than other GPU systems, and even than CPU system for H-size images.

Chapter 5

Approach for Latency Hidden

5.1 Algorithm Overview

This algorithm is based on the Domain Transformation described in section 2.2.3. As a global algorithm, it has been widely used by many researchers to improve the accuracy of their systems and achieved good results. However, this 4-way cost propagation method has the difficulty for achieving high processing speed, because the costs of each pixel has to be calculated one by one to propagate its cost to its neighbor pixels. In this chapter, we aim to construct a real-time stereo vision system based on this global algorithm by using the GPUs. Not only with the high-end GPU used on PC, we also implemented this system with an embedded GPU to check its effects in the real world environment.

In order to save the execute time as much as possible, we use the simple cost calculation method *Census-Transform* described in Section 2.1.1. In addition, the SMP GCP detection method described in Section 2.3.1 is also used here. Thus, different from the previous two methods, we only need to use the left image as the base, and do the stereo matching one time.

5.1.1 Processing Flow

The input images are processed as follows.

1. the two input images are gray-scaled,
2. the two images are scaled down,
3. the Census-Transform (CT) is calculated as the matching cost of each pixel,
4. cost aggregation from left to right along the x-axes.
5. cost aggregation along the y-axes.

6. cost aggregation from right to left along the x-axes.
7. Detecting GCPs while generating the disparity map.
8. refinement by a median filter and filling the non-GCPs by using a bilateral estimation method
9. scaling up the disparity map

Here, the processing flow is basically the same as the approach described in chapter 3 except some adjustments in the order of cost aggregation. In this system, the aggregation from right to left along the x-axes is done later, because it can be synchronized with SMP.

5.2 Implementation on GPU

5.2.1 System Pipeline

For most of the processing steps, we followed the previous approaches. In this section, we focus on the cost aggregation step. Fig.5.1 shows the task assignment of the Domain Transformation aggregation. Different from the previous methods, the pixels are not processed in parallal, but the different disparities are processed in parallel. In *Step*1, for each SM, D threads are assigned to calculate the matching cost $C_L(x, y, d)$ for each pixel in parallel. According to (2.16), each $C_L(x, y, d)$ is aggregated one by one along the x-axis, and it is transferred to the global memory every time. The intermediate result $C_L(x-1, y, d)$ is propagated through the on-chip register. In *Step*2, the cost $C_L(x, y, d)$ is aggregated along the y-axis, and each SM will be transposed to be processed column by column. Although the method described in Section 4.2.3 can be used here, it requires access to the global memory twice for each calculation. It is not suitable for the embedded GPU which has a large access latency. Therefore, we use the registers to create a buffer, which follows the rules of FIFO. The buffer is used to store the costs which are used during the aggregation. Its size is fixed at W as mentioned in (2.20). The *Step*3 is basically the same as *Step*1, except that it no longer needs to transfer the cost back to the global memory, but does the WTA directly. Here, because the disparity map is generated along the x-axis, the SMP can also be done at the same time.

5.2.2 Latency Hidden

Because the Domain Transformation approach requires access to the global memory frequently, hiding the latency is the most important point for the acceleration. To hide the latency of the global memory, two methods can be considered:

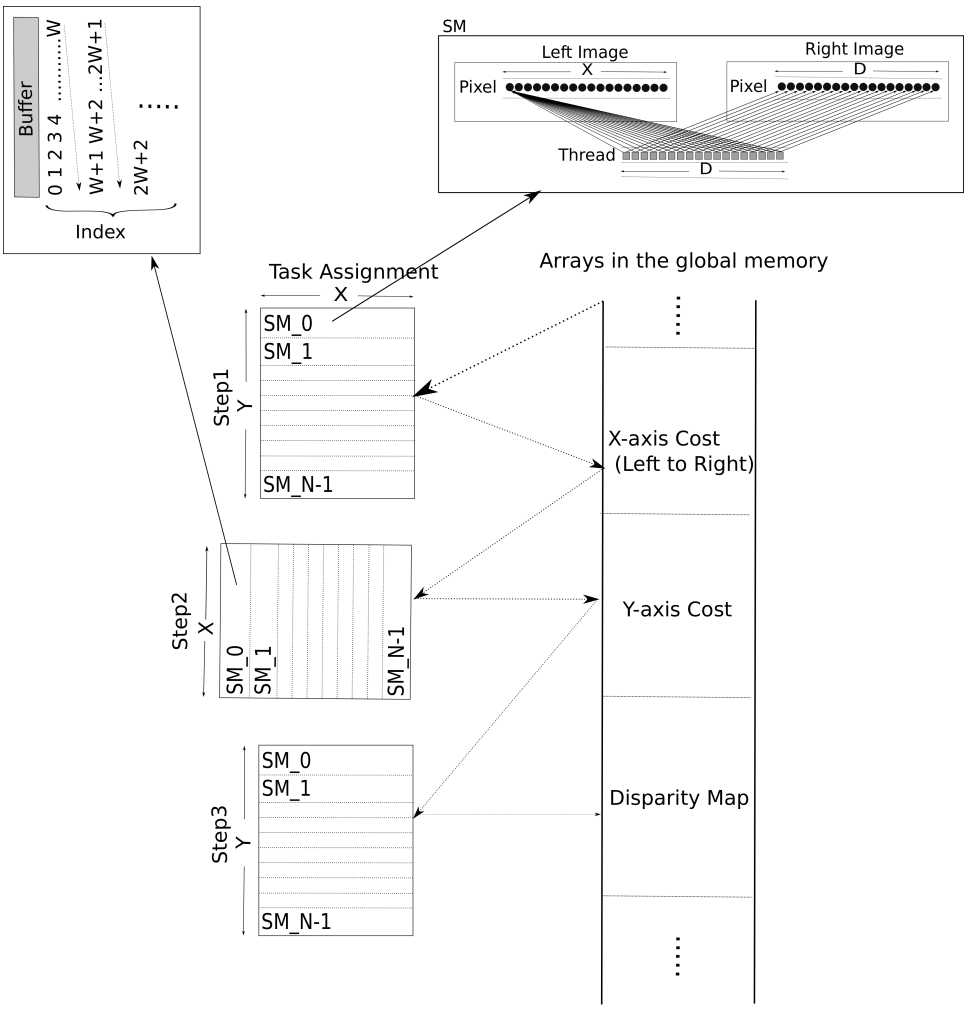

Figure 5.1: Task assignment to each step on GPU. *Step1*: Cost Aggregation from left to right along the x-axis. *Step2*: Cost Aggregation along the y-axis. *Step3*: Cost Aggregation from right to left, WTA & SMP.

Table 5.1: Parameters to access single precision data

	Jetson TX2
Architecture	Pascal
Global Memory Latency (clock cycles)	1051
Bandwidth (Gb/s)	59.7
Base clock cycle (MHz)	1300
SM	2
Transmission Rate (Byte/clock cycle)	46
Data Requirement (bytes)	48,356
Data Requirement/SM (bytes)	24,178
Flops/clock cycle/core	2

1. keep the operation units busy by executing more than N_{FMA} operations (the lowest value to make the units busy) in each SM for the current data set until the next data set arrive from the global memory by data prefetching, and

2. transfer a large volume of data (V_s) from the global memory continuously.

Although many kinds of operations can be used for the Domain Transformation, we mainly use the Fused Multiply-Add(FMA) units in this system. In most cases, the first approach is preferable, because the data loading overhead from the global memory can be relatively reduced more by executing more number of operations per loaded data. However, in the Domain Transformation, the amount of cost data of each pixel is usually too small for this first approach, resulting in less number of executable operations. Thus, the second approach is also required. Here, it is necessary to make it clear under what conditions which method shows better performance.

Table 5.1 shows several parameters of Jetson TX2 GPU and its performance for accessing single precision data. As shown in Table 5.1, in Jetson TX2, 2 FMA operations can be executed in one clock cycle in each core, namely 256 FMA operations in each SM (each SM has $N_{cores} = 128$ cores). According to the method proposed in [17], the global memory latency of Jetson TX2 is 1051 clock cycles. In order to hide this 1051 clock cycles, $N_{FMA} = 269,056$ FMA operations ($269,056 = 1051 \times N_{cores} \times 2$) are required in each SM for the current data set.

The volume size V_s can be calculated as follows. The Jetson TX2 has a base clock of 1300 MHz and the bandwidth of 59.7 GB/s, which means that the transfer rate is roughly 46 bytes per clock cycle. Therefore, the volume size which is required to hide the latency (1051 clock cycles) becomes 48,346 = 46 $\times$ 1051 bytes. This means that the minimum volume size to make the global memory busy is V_s = 48,356 bytes. For dividing the data, and assigning them to each SM,

the following procedure should be taken:

1. Evaluate the number of FMA operations that can be executed for the data in each SM.

2. If it is larger than N_{FMA}, use the the first method which is based on the data prefetching. If not, redivide the data so that the total size of data that are transferred to all SMs becomes larger than V_s, and use the second approach.

In the Domain Transformation, the average number of operators for each pixel is roughly 10. If we use the method 1, assuming the range of disparity is 128 and each SM processes 8 lines in each time, the number of pixels which we need to hide the latency is $N_{pixel} = N_{FMA}/(128 \times 8lines \times 2SMs \times 10) \approx 13$. On the other hand, if we use the method 2, the number of pixels which we need to hide the latency is $N_{pixel} = V_s/(128 \times 8lines \times 4bytes \times 2SMs) \approx 6$.

By comparing them, it can be known that method 2 is better than method 1, because it requires less number of registers which leads to the improvement of overall degree of parallelism. Therefore, the method 2 is used in our system to hide the latency.

5.3 Experimental Results

We have implemented the algorithm on an embedded GPU Jetson Tx2 and a High-End GPU Nvidia GTX 1080Ti repsectively. In this system, we not only evaluated the processing speed and the error rate using the KITTI 2015 [27] benchmark, but also run it in the real world environment.

Here, since the numbers of lines in KITTI are small (1250×375), we only scaled down the images along the x-axis (K=2). According to our evaluation, the runtime of our system on Jetson TX2 is roughly 0.018s (55.6fps) and is roughly 0.00154s (646fps) on GTX 1080Ti. In KITTI, two data-sets, training data-set and testing data-set, are provided, and the true depth maps only for the training data-set are given. Users tune up their algorithms using the training data-set, and upload their results of the testing data-set to KITTI web-page, and the algorithms are evaluated and ranked. Our accuracy for the testing data-set is shown in Table.5.2. As for the accuracy of the training data-set, we achieved the lowest error rate 2.7% for "KITTI2015-000147", while the worst error rate 70% for "KITTI2015-000104" as shown in Fig.5.2. This accuracy is not good due to the limitations of the algorithm, but it is already far faster than other stereo systems.

Fig.5.3 shows the matching results of the images taken by ZED stereo camera, and our system in the real world environment. Here, the size of the input images is 1280×720 and the range of disparity is set as 128. This comparison shows that our results are better than the ZED stereo camera visually, though it is not easy to

Table 5.2: KITTI2015

Error	D1-bg	D1-fg	D1-all
All / All	11.55 %	19.69 %	12.91 %
All / Est	10.55 %	19.58 %	12.03 %
Noc / All	10.55 %	18.90 %	12.77 %
Noc / Est	9.47 %	18.80 %	11.86%

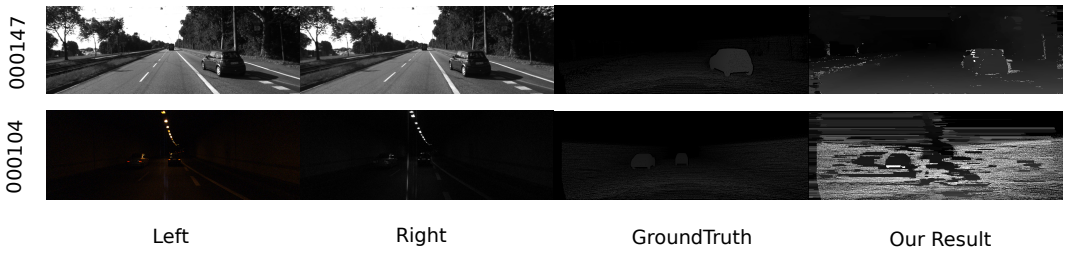

Figure 5.2: Matching Result(KITTI2015)

Figure 5.3: Matching Result(Zed)

compare precisely because the true depth cannot be known. Our implementation is very simple, and its processing speed is 23 fps even when the images are not scaled down.

Chapter 6

General Discussion

Our first approach (chapter 3) implemented the local stereo matching algorithm based on Cost Aggregation for high resolution images on a GTX 780Ti GPU. In this approach, we tried to reduce the total amount of computation by scaling down the images and reusing the intermediate matching results. In addition, we also focused on the ways to avoid the bank conflict of shared memory. To comparison with other fast stereo matching systems, it can be known that the processing speed of our system is much faster than previous ones, and its accuracy is also not bad according to our evaluation by using the Middlebury Benchmark V3. These facts mean that our approach is very suitable for accelerating the stereo matching for high resolution images. However, the matching accuracy is still limited owing to the scaling-down of the input images.

Our second approach (chapter 4) implemented the local stereo matching algorithm based on Multi-Block Matching (MBM) for high resolution images on two GPUs with different architecture. According to the evaluation results on two Benchmark sets (Middlebury and KITTI), our system is not only fast, but also maintains the original accuracy of MBM algorithm. Compared with other approaches including our first approach, our second approach is faster and more precise, mainly due to our effective acceleration of MBM and the investment of secondary matching. The acceleration method was originally designed for GTX 780 Ti, but it could be easily ported on GTX 1080 Ti, and showed good performance on the two different GPU architectures. The performance gain by new architecture is 3X, and this is that we can expect from the difference of their maximum throughput. This means that it can be expected that our method can also achieve higher performance on upcoming new GPUs.

Our third approach (chapter 5) implemented the global stereo matching algorithm based on Domain Transformation for high resolution images on an embedded GPU. Different from the local algorithm, due to the high dependency between the data in global algorithm, it cannot be parallelized easily. Hence, we have to pro-

cess the pixel one by one and to access the global memory frequently. Fortunately, the synchronization between different disparities allows us to parallelize the algorithm from another perspective, and we found an effective way to hide the latency caused by memory accessing. This method can help us use different methods to hide latency under different conditions. According our experimental results, this method is very effective in accelerating the calculation of convolution in CNN, even exceeding the standard library. By using it in the stereo matching, plus the approaches proposed previously, the processing speed of our system has been greatly improved. Compared with the previous approaches, this is the fastest matching system implemented by us. We also evaluated our system by using the KITTI2015 Benchmark. Here, due to the amount of the computation, we only chose the simplest Census Transform and SMP to combine with the Domain Transformation, resulting in an unsatisfactory accuracy. In fact, when we choose other methods (like CNN) to combine with Domain Transformation, the accuracy of our system can be greatly improved, of course it also takes more time. To further evaluate this system, we first run it in a real world by using the combination of Jetson TX2 and Zed stereo camera. Compared with the sample program supported by Zed, our results is clearer and is enough for most practical use.

In our research, we proposed different acceleration approaches for different algorithms, mainly because different algorithms have different bottlenecks. However, the approaches proposed by us can be combined and applied to various stereo matching systems. Scaling down a set of matching images can obviously reduce the amount of calculation and improve the processing speed of matching. The factor of scaling can be defined by the users according to their requirements. Generally, it will not cause too much loss of accuracy for high resolution images because of a large amount of information of high resolution images. However, there are still many situations that require more precise matching. In these cases, secondary matching is required. The secondary matching is a process to fine-tune disparities, and it helps us to find a more precise matching result in a limited space using limited information. Therefore, it does not require too much time or memory space, and works well with the scaling down approach. The latency hidden method helps us to clarify the key to hide the memory access latency under different conditions so that we can choose the most efficient way. It can be used not only in global algorithm, but also in local algorithm such as the method described in Section 4.2.3, even the algorithms other than stereo matching. Our approaches are not specific to a particular GPU, and they can be ported easily to other GPU systems, even to other platforms such as FPGAs.

Chapter 7

Conclusions and Future Directions

In this research we proposed and implemented a real-time stereo matching approach for high resolution images. This approach consists of three different parts, which represent the study of acceleration methods from three different perspective. We studied them independently and implemented them on three different stereo systems with three different stereo matching algorithms: Cross-Aggregation (Section 2.2.1), Multi-Block Matching (Section 2.2.2) and Domain Transformation (Section 2.2.3). Each algorithm is used in combination with other different matching cost calculation algorithms (Section 2.1) and disparity refinement algorithms (Section 2.3). In each system, we not only proposed a general method to increase the performance of stereo matching for high resolution images, but also proposed a specific solution for each aggregation algorithm.

We run our systems on different GPUs and evaluated the processing speed and the accuracy of them under different conditions. According to the evaluation results, it can be known that although the accuracy of our system is not very stable, the processing speed is always far faster than other systems. In terms of processing speed, the third system is the fastest among our three systems. The processing speed of third system has achieved 646fps on GTX 1080Ti GPU for the image sets of KITTI 2015 Benchmark, while the second one achieved 66fps on GTX 780Ti GPU. Similarly, in terms of accuracy, our second system which focus on improving the accuracy achieved the best precision. All of these shows that our approach works very well.

7.1 Contributions of this Work

The main contributions of this work concern with a fast stereo matching system for high resolution images. First, to the best of our knowledge, we first presented

the GPU acceleration method of real-time stereo matching for high resolution images. The second one is that we presented a accuracy improvement method to prevent the loss of precision caused by the first method. The third one is that we presented a latency hidden method which can help us avoid the limitations caused the latency and implement more complex algorithms efficiently.

In short, for the first time in the literature, an GPU stereo matching approach that covers both the high processing speed and high accuracy for high resolution images has been proposed. Thanks to the parallel and pipeline processing, our approach achieves an ideal throughput for both the local and global algorithms. In addition, our approach can be easily combined with other algorithms which can improve the accuracy of system. Furthermore, it can easily be implemented on other GPUs, even on other platform.

7.2 Future Directions

So far, all of our efforts are to maintain the accuracy of the original algorithm as much as possible, resulting in a limited precision. In the future, our approach can be combined with other more complex algorithms such as CNN to improve the accuracy of our system. Certainly, more in-depth acceleration methods are needed for the entire calculation. According to our evaluations, if we only use one GTX 1080Ti to accelerate this calculation, the processing cannot be achieved in real-time. But, it is possible to use multiple GPUs or higher performance GPUs in future.

Moreover, during the scaling up step, we used the Bilateral Estimation method to fill the disparity map. However, this method limits the improvement of accuracy. Recently, with the rapid development of deep learning, the research on super resolution convolution neural network (SRCNN) is growing. It is mainly studied to enlarge small images precisely, and it can be used to scale up the disparity map in our system. This is one of our future work.

References

[1] Daolu Zha, Xi Jin, Tian Xiang: *A real-time global stereo-matching on FPGA*. Microprocessors and Microsystems - Embedded Hardware Design 47: 419-428 (2016)

[2] Madaín Pérez Patricio and Abiel Aguilar-González and Miguel O. Arias-Estrada and Héctor-Ricardo Hernandez-de Leon and Jorge-Luis Camas-Anzueto and J. A. de Jesús Osuna-Coutiño: *An FPGA stereo matching unit based on fuzzy logic*. Microprocessors and Microsystems - Embedded Hardware Design 42: 87-99 (2016)

[3] Alex Kendall, Hayk Martirosyan, Saumitro Dasgupta, Peter Henry, Ryan Kennedy, Abraham Bachrach, Adam Bry: *End-to-End Learning of Geometry and Context for Deep Stereo Regression*. CoRR abs/1703.04309 (2017)

[4] Wenbao Qiao and Jean-Charles Créput: *Stereo Matching by Using Self-distributed Segmentation and Massively Parallel GPU Computing*. ICAISC (2) 2016: 723-733

[5] Jure Zbontar, Yann LeCun: *Stereo Matching by Training a Convolutional Neural Network to Compare Image Patches*. Journal of Machine Learning Research 17: 65:1-65:32 (2016)

[6] Leonid Keselman, John Iselin Woodfill, Anders Grunnet-Jepsen, Achintya Bhowmik: *Intel RealSense Stereoscopic Depth Cameras*. CoRR abs/1705.05548 (2017)

[7] Mohammad Dehnavi, Mohammad Eshghi: *FPGA based real-time on-road stereo vision system*. Journal of Systems Architecture - Embedded Systems Design 81: 32-43 (2017)

[8] Luca Puglia, Mario Vigliar, Giancarlo Raiconi: *Real-Time Low-Power FPGA Architecture for Stereo Vision*. IEEE Trans. on Circuits and Systems 64-II(11): 1307-1311 (2017)

[9] Nils Einecke, Julian Eggert: *A multi-block-matching approach for stereo.* Intelligent Vehicles Symposium 2015: 585-592

[10] Andrey Kuzmin, Dmitry Mikushin, Victor S. Lempitsky: *End-to-end Learning of Cost-Volume Aggregation for Real-time Dense Stereo.* CoRR abs/1611.05689 (2016)

[11] Daniel Hernández Juárez and Alejandro Chacón and Antonio Espinosa and David Vázquez and Juan Carlos Moure and Antonio M. López: *Embedded Real-time Stereo Estimation via Semi-Global Matching on the GPU.* ICCS 2016: 143-153

[12] Minxi Jin, Tsutomu Maruyama: *Fast and Accurate Stereo Vision System on FPGA.* TRETS 7(1): 3:1-3:24 (2014)

[13] Jedrzej Kowalczuk and Eric Psota and Lance C. Pérez: *Real-Time Stereo Matching on CUDA Using an Iterative Refinement Method for Adaptive Support-Weight Correspondences.* IEEE Trans. Circuits Syst. Video Techn. 23(1): 94-104 (2013)

[14] Qingxiong Yang and Liang Wang and Ruigang Yang and Henrik Stewénius and David Nistér: *Stereo Matching with Color-Weighted Correlation, Hierarchical Belief Propagation, and Occlusion Handling.* IEEE Trans. Pattern Anal. Mach. Intell. 31(3): 492-504 (2009)

[15] Heiko Hirschmüller and Daniel Scharstein: *Evaluation of Cost Functions for Stereo Matching.* CVPR 2007

[16] D.Scharstein and R.Szeliski: *http://vision.middlebury.edu/stereo/eval3/.*

[17] X. Mei, X. Chu, *Dissecting GPU memory hierarchy through microbenchmarking,* IEEE Trans. Parallel Distrib. Syst, 2016.

[18] Stepan Tulyakov, Anton Ivanov, François Fleuret: *Weakly Supervised Learning of Deep Metrics for Stereo Reconstruction.* ICCV 2017: 1348-1357

[19] Heiko Hirschmüller: *Accurate and Efficient Stereo Processing by Semi-Global Matching and Mutual Information.* CVPR (2) 2005: 807-814

[20] Nils Einecke, Julian Eggert: *A Two-Stage Correlation Method for Stereoscopic Depth Estimation.* DICTA 2010: 227-234

[21] Andreas Geiger, Martin Roser, Raquel Urtasun:*Efficient Large-Scale Stereo Matching.* ACCV (1) 2010: 25-38

[22] Song Zhang, Peisen Huang: *High-Resolution, Real-time 3D Shape Acquisition.* CVPR Workshops 2004: 28

[23] H. Nguyen, D. Nguyen, Z. Wang, H. Kieu, and M. Le: *Real-time, high accuracy 3d imaging and shape measurement.* Appl. Opt. 54, A9A17 (2015).

[24] Xióngbiao Luó and Uditha L. Jayarathne and Stephen E. Pautler and Terry M. Peters: *Binocular Endoscopic 3-D Scene Reconstruction Using Color and Gradient-Boosted Aggregation Stereo Matching for Robotic Surgery.* ICIG (1) 2015: 664-676

[25] Cuong Cao Pham, Vinh Quang Dinh, Jae Wook Jeon: *Robust non-local stereo matching for outdoor driving images using segment-simple-tree.* Sig. Proc.: Image Comm. 39: 173-184 (2015)

[26] Andreas Geiger, Philip Lenz, Raquel Urtasun: *Are we ready for autonomous driving? The KITTI vision benchmark suite.* CVPR 2012: 3354-3361

[27] Moritz Menze, Andreas Geiger: *Object scene flow for autonomous vehicles.*

[28] Philip J. Fleming, John J. Wallace: *How Not To Lie With Statistics: The Correct Way To Summarize Benchmark Results.* Commun. ACM 29(3): 218-221 (1986)

[29] Zhengfa Liang, Yiliu Feng, Yulan Guo, Hengzhu Liu, Wei Chen, Linbo Qiao, Li Zhou, Jianfeng Zhang: *Learning for Disparity Estimation through Feature Constancy.* CoRR abs/1712.01039 (2018)

[30] C. Stentoumis, L. Grammatikopoulos, I. Kalisperakis, and G. Karras: *On accurate dense stereo-matching using a local adptive multi-cost approach.* ISPRS J. Photogram. Remote Sens., vol. 91, pp. 29-49, May 2014.

[31] Simon Hadfield, Karel Lebeda, Richard Bowden: *Stereo reconstruction using top-down cues.* Computer Vision and Image Understanding 157: 206-222 (2017)

[32] Ning Ma, Yubo Men, Chaoguang Men, Xiang Li: *Accurate Dense Stereo Matching Based on Image Segmentation Using an Adaptive Multi-Cost Approach.* Symmetry 8(12): 159 (2016)

[33] Kang Zhang, Jiyang Li, Yijing Li, WeiDong Hu, Lifeng Sun, Shiqiang Yang: *Binary stereo matching.* ICPR 2012: 356-359

[34] Konstantinos Batsos, Changjiang Cai, Philippos Mordohai:*CBMV: A Coalesced Bidirectional Matching Volume for Disparity Estimation.* CoRR abs/1804.01967 (2018)

[35] Ang Li, Dapeng Chen, Yuanliu Liu, Zejian Yuan: *Coordinating Multiple Disparity Proposals for Stereo Computation.* CVPR 2016: 4022-4030

[36] Haihua Lu, Hai Xu, Li Zhang, Yong Zhao: *Cascaded multi-scale and multi-dimension convolutional neural network for stereo matching.* CoRR abs/1803.09437 (2018)

[37] Chi Zhang, Zhiwei Li, Yanhua Cheng, Rui Cai, Hongyang Chao, Yong Rui: *MeshStereo: A Global Stereo Model with Mesh Alignment Regularization for View Interpolation.* ICCV 2015: 2057-2065

[38] Jia-Ren Chang, Yong-Sheng Chen: *Pyramid Stereo Matching Network.* CoRR abs/1803.08669 (2018)

[39] Cevahir Cigla: *Recursive edge-aware filters for stereo matching.* CVPR Workshops 2015: 27-34

[40] Rostam Affendi Hamzah, Haidi Ibrahim, Anwar Hasni Abu Hassan: *Stereo matching algorithm based on per pixel difference adjustment, iterative guided filter and graph segmentation.* J. Visual Communication and Image Representation 42: 145-160 (2017)

[41] L. Li, X. Yu, S. Zhang, X. Zhao, and L. Zhang: *3D cost aggregation with multiple minimum spanning trees for stereo matching.* Applied Optics, vol. 56, no. 12, pp. 3411-3420 (2017).

[42] Stefan K. Gehrig, Reto Stalder, Nicolai Schneider: *A Flexible High-Resolution Real-Time Low-Power Stereo Vision Engine.* ICVS 2015: 69-79

[43] Wenqiang Wang, Jing Yan, Ningyi Xu, Yu Wang, Feng-Hsiung Hsu: *Real-Time High-Quality Stereo Vision System in FPGA. IEEE Trans. Circuits Syst.* Video Techn. 25(10): 1696-1708 (2015)

[44] Soenke Michalik, Soeren Michalik, Jamin Naghmouchi, Mladen Berekovic: *Real-time smart stereo camera based on FPGA-SoC.* Humanoids 2017: 311-317

[45] Kyeong-Ryeol Bae, Byungin Moon: *An accurate and cost-effective stereo matching algorithm and processor for real-time embedded multimedia systems.* Multimedia Tools Appl. 76(17): 17907-17922 (2017)

[46] Liang Wang, Ruigang Yang, Minglun Gong, Miao Liao: *Real-time stereo using approximated joint bilateral filtering and dynamic programming.* J. Real-Time Image Processing 9(3): 447-461 (2014)

[47] Simone Madeo and Riccardo Pelliccia and Claudio Salvadori and Jesús Martínez del Rincón and Jean-Christophe Nebel: *An optimized stereo vision implementation for embedded systems: application to RGB and infra-red images.* J. Real-Time Image Processing 12(4): 725-746 (2016)

[48] Christos Ttofis, Theocharis Theocharides: *High-quality real-time hardware stereo matching based on guided image filtering.* DATE 2014: 1-6

[49] Minh Nguyen, Wei Qi Yan, Rui Gong, Patrice Delmas: *Toward a real-time belief propagation stereo reconstruction for computers, robots, and beyond.* IVCNZ 2015: 1-6

[50] Vaddi Chandra Sekhar, Satyajit Bora, Monalisa Dash, Manchi Pavan Kumar, S. Josephine, Roy Paily: *Design and Implementation of Blind Assistance System Using Real Time Stereo Vision Algorithms.* VLSI Design 2016: 421-426

[51] Kyuho Jason Lee, Kyeongryeol Bong, Changhyeon Kim, Junyoung Park, Hoi-Jun Yoo: *An energy-efficient parallel multi-core ADAS processor with robust visual attention and workload-prediction DVFS for real-time HD stereo stream.* COOL Chips 2016: 1-3

[52] Jaco Hofmann, Jens Korinth, Andreas Koch: *A Scalable High-Performance Hardware Architecture for Real-Time Stereo Vision by Semi-Global Matching.* CVPR Workshops 2016: 845-853

[53] Lucas F. S. Cambuim and João Paulo Fernandes Barbosa and Edna Natividade da Silva Barros: *Hardware module for low-resource and real-time stereo vision engine using semi-global matching approach.* SBCCI 2017: 53-58

[54] Chen Yang, Yan Li, Wei Zhong, Song Chen: *Real-Time Hardware Stereo Matching Using Guided Image Filter.* ACM Great Lakes Symposium on VLSI 2016: 105-108

[55] Ratheesh Kalarot, John Morris: *Comparison of FPGA and GPU implementations of real-time stereo vision.* CVPR Workshops 2010: 9-15

[56] Fouzhan Hosseini, Amir Fijany, Saeed Safari, Ryad Chellali, Jean-Guy Fontaine: *Real-Time Parallel Implementation of SSD Stereo Vision Algorithm on CSX SIMD Architecture.* ISVC (1) 2009: 808-818

[57] Christian Zinner, Martin Humenberger, Kristian Ambrosch, Wilfried Kubinger: *An Optimized Software-Based Implementation of a Census-Based Stereo Matching Algorithm.* ISVC (1) 2008: 216-227

[58] Nils Einecke, Julian Eggert: *A Two-Stage Correlation Method for Stereoscopic Depth Estimation.* DICTA 2010: 227-234

[59] Federico Tombari, Stefano Mattoccia, Luigi di Stefano: *Full-Search-Equivalent Pattern Matching with Incremental Dissimilarity Approximations.* IEEE Trans. Pattern Anal. Mach. Intell. 31(1): 129-141 (2009)

[60] Luigi di Stefano, Massimiliano Marchionni, Stefano Mattoccia: *A fast area-based stereo matching algorithm.* Image Vision Comput. 22(12): 983-1005 (2004)

[61] S Mattoccia: *A locally global approach to stereo correspondence.* Computer Vision Workshops (ICCV Workshops), 2009.

[62] Z. Wang and Z. Zheng, *A region based stereo matching algorithm using cooperative optimization.* CVPR 2008

[63] M. A. Fischler and R. C. Bolles, Random Sample Consensus: *A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography.* Comm. of the ACM 24: 381395, June 1981

[64] Carlo Tomasi, Roberto Manduchi: *Bilateral Filtering for Gray and Color Images.* ICCV 1998: 839-846

[65] Qiong Chang, Tsutomu Maruyama: *Real-Time Stereo Vision System: A Multi-Block Matching on GPU.* IEEE Access 6: 42030-42046 (2018)

[66] Qiong Chang, Tsutomu Maruyama: *Real-Time High-Quality Stereo Matching System on a GPU.* ASAP 2018: 1-8

[67] Qiong Chang, Masaki Onishi, Tsutomu Maruyama: *Fast convolution kernels on pascal GPU with high memory efficiency.* SpringSim (HPC) 2018: 3:1-3:12

[68] Cuong Cao Pham, Jae Wook Jeon: *Domain Transformation-Based Efficient Cost Aggregation for Local Stereo Matching.* IEEE Trans. Circuits Syst. Video Techn. 23(7): 1119-1130 (2013)

[69] Lu Zhang, Ke Zhang, Tian Sheuan Chang, Gauthier Lafruit, Georgi Krasimirov Kuzmanov, Diederik Verkest: *Real-time high-definition stereo matching on FPGA.* FPGA 2011: 55-64

[70] Ke Zhang, Jiangbo Lu, Gauthier Lafruit: *Cross-Based Local Stereo Matching Using Orthogonal Integral Images.* IEEE Trans. Circuits Syst. Video Techn. 19(7): 1073-1079 (2009)

[71] Edoardo Paone, Gianluca Palermo, Vittorio Zaccaria, Cristina Silvano, Diego Melpignano, Germain Haugou, Thierry Lepley: *An exploration methodology for a customizable OpenCL stereo-matching application targeted to an industrial multi-cluster architecture.* CODES+ISSS 2012: 503-512

Research Achievements

Journals (First author)

- Qiong Chang, Tsutomu Maruyama: *Real-Time Stereo Vision System: A Multi-Block Matching on GPU.* IEEE Access 6: 42030-42046 (2018)

International Conference Papers (First author)

- Qiong Chang, Tsutomu Maruyama: *Real-Time High-Quality Stereo Matching System on a GPU.* ASAP 2018: 1-8
- Qiong Chang, Masaki Onishi, Tsutomu Maruyama: *Fast convolution kernels on pascal GPU with high memory efficiency.* SpringSim (HPC) 2018: 3:1-3:12